Federico Giannini

CORSO DI PHP

Livello base. Con esercizi

© 2014 Losna (www.losnaweb.com) – PI 01213920455, REA 122436 CCIAA Massa Carrara

Federico Giannini (Massa, 1986) è webdesigner e webdeveloper professionista.

Introduzione a PHP

PHP. Una sigla con la quale tutti noi che navighiamo in rete abbiamo avuto a che fare qualche volta (anche semplicemente per aver trovato una pagina con il suffisso **".php"**). Una sigla che per qualcuno evoca un obiettivo da raggiungere nell'ambito dello **sviluppo di siti web**. Che per qualcun altro evoca uno dei primi CMS, quel famoso **"PHP-Nuke"** tanto in voga qualche anno fa. Che per i miei ex compagni di università evoca notti insonni passati davanti al computer per passare un esame… ! Ma alla fine una sigla che per noi rappresenterà un modo per creare **siti dinamici**.

Ecco, mettiamoci subito d'accordo sul **termine "dinamico"**. Capita spesso (è capitato anche a me) di avere clienti che chiedono un sito "dinamico", e con "dinamico" il 99% delle volte intendono un sito "che si muove" (con animazioni in Flash, per intenderci). **Non è affatto così**. E per capire cosa si intende correttamente per "dinamico", dobbiamo sapere che il **linguaggio PHP** è un linguaggio **lato server**, a differenza, per esempio, di JavaScript che invece è un linguaggio **lato client**.

Cosa significa questo in breve? Significa che uno script JavaScript verrà **interpretato dal vostro browser**, che eseguirà il codice e si comporterà di conseguenza. Invece le operazioni di un linguaggio **lato server come PHP** saranno compiute, come dice il termine stesso, **dal server**, che consegnerà al client (e cioè al browser nel nostro caso) la **pagina HTML**, codificata dopo aver eseguito una serie di operazioni.

Avete mai provato a vedere il **codice delle pagine PHP** sulle quali navigate? Bene, avrete sicuramente notato che se cliccate **"sorgente pagina"** (io uso Firefox) su una pagina di un sito in PHP vedrete che in realtà non c'è **neppure una riga di codice PHP**, ma soltanto HTML. Se invece osservate il codice di una pagina che ha uno script JavaScript, vedrete al contrario lo script nella sua interezza. Perché succede questo? Molto semplice: il codice PHP sta **sul server**, che lo interpreta e, come detto prima, consegna al browser la pagina già codificata. Per questo motivo vedete solo HTML e non PHP. Quindi, non è neppure possibile **copiare il codice** PHP (una volta che è stato eseguito) da un altro sito.

Tornando quindi al termine **"dinamico"**, ora possiamo dire che i linguaggi lato server producono **dinamicamente** il contenuto a seconda delle richieste dell'utente, anche con **interrogazioni al database** (l'**interazione tra PHP e un database di tipo MySQL** è uno degli argomenti di questo corso). Ecco cosa si intende per "dinamico"! Pensiamo per esempio a un sito che ci serve per cercare **un dato** in un **database** con centinaia di campi: per esempio un indirizzo di un ristorante in un sito che ospita un database di tutti i ristoranti d'Italia. Noi cercheremo un ristorante in Liguria (magari attraverso un semplice **form di ricerca**), e il server, interpretando la nostra richiesta, ci produrrà **in modo dinamico** una pagina che conterrà gli indirizzi di tutti i ristoranti liguri memorizzati nel database, tralasciando quelli che non ci interessano. Invece intendiamo con **"statico"** un

sito prodotto con un linguaggio come HTML: esiste una sola pagina ed è quella per tutti, e non ci sono contenuti che vengono prodotti a seconda delle richieste dell'utente.

Ma torniamo al punto da cui siamo partiti: la sigla. Cosa significa PHP? È un cosiddetto acronimo **"ricorsivo"**, perché significa **"PHP Hypertext Preprocessor"** ("PHP Preprocessore di Ipertesti"): un acronimo ricorsivo è un acronimo in cui una delle lettere sta per la sigla, insomma un acronimo che contiene se stesso. **"Preprocessore"** è invece un termine informatico con il quale ci si riferisce a un programma che compie alcune **operazioni preliminari** prima dell'esecuzione del codice: il nostro linguaggio contiene il termine "preprocessore" perché prima di inviare la pagina HTML finale all'utente vengono interpretate, appunto, le istruzioni del codice, quindi prima che la pagina HTML venga consegnata al nostro browser, il preprocessore PHP compie alcune operazioni.

Che **espressione** utilizzeremo per indicare PHP? O meglio… che **tipo di linguaggio** è? Non è ovviamente un **linguaggio di markup**. Il papà di PHP, il danese nato in Groenlandia **Rasmus Lerdorf**, lo chiama **"linguaggio di scripting"**, quindi io direi di utilizzare questa espressione, che sta bene anche con JavaScript. Curiosità: PHP è stato inventato nel 1994. Da allora ne ha fatta di strada!

I **vantaggi** che derivano dall'**utilizzo di PHP** sono davvero molti e li scopriremo man mano che andremo avanti con il corso. Possiamo anticipare che PHP è facile da apprendere (certo, ci vogliono un po' di pratica e un po' di esercizio), che PHP ci semplificherà notevolmente la vita per alcune operazioni, che grazie a PHP potremo gestire meglio i dati del nostro sito, e che con PHP potremo anche, se avremo pazienza e soprattutto tempo a disposizione, costruire un nostro piccolo e semplice CMS personale!

Questo corso è stato pensato per chi non ha mai scritto in vita sua **una sola riga di codice PHP**, e il suo punto di forza è la **chiarezza espositiva**: gli argomenti di cui parleremo saranno infatti sempre corredati di **esempi**, ognuno con il proprio codice. E per fissare meglio i concetti appresi, ho pensato anche ad alcuni piccoli esercizi che ho messo a conclusione dei capitoli più importanti. Tuttavia, per poter intraprendere questo corso è necessaria una minima infarinatura di **HTML** perché ci troveremo spesso a manipolare, appunto, codice HTML attraverso PHP.

E ora, non ci resta altro che avventurarci nei meandri della programmazione in PHP!

1. Creare siti in PHP: cosa ci serve per iniziare

Prima di iniziare a mettere le mani sul codice, è necessario sapere che creare un sito con PHP (alcuni dicono anche “programmare in PHP”!) **è un po’ diverso** rispetto a creare un sito con HTML e CSS, perché abbiamo bisogno di qualche **strumento aggiuntivo**, necessario per poter svolgere il nostro compito.

Quello che ci serve è un cosiddetto “**ambiente di sviluppo”** (o “**piattaforma di sviluppo”**, se preferite): dal momento che PHP è una tecnologia **lato server**, dovremo creare un piccolo server sul nostro computer in modo tale che il nostro codice possa funzionare, e insieme a questo server dovremo avere anche un programma che ci permetterà di gestire il database, se vorremo utilizzarlo.

La scelta è vivamente consigliata, perché se iniziate a mettere le mani sul PHP, prima o poi vi capiterà sicuramente di dover creare un sito **PHP/MySQL**. Ma di questo parleremo più avanti! E infine, ovviamente, dovremo avere installato sul nostro computer, oltre a server e database, il codice.

Queste **tre componenti** (codice, server e database) costituiscono l’**ambiente di sviluppo**. Sono davvero molti gli ambienti di sviluppo che si trovano in giro per la rete, ma io vi consiglio **WAMP**, acronimo di **W**indows (tra poco vi dirò come fare con Linux), **A**pache (il nome del server), **M**ySQL (il database: il programma integrato in WAMP per la gestione dei database è **PhpMyAdmin**) e, naturalmente, **P**hp. Per scaricare WAMP basta recarsi sul sito www.wampserver.com (o www.wampserver.com/en, se non avete familiarità con il francese).

Dicevo, per gli utenti che utilizzano Linux, l’ambiente di sviluppo (server, PHP e MySQL: potete chiamarlo “Lamp”, con la “L” che sta per “Linux”) si trova già nel sistema operativo e basterà avviarlo (vi consiglio però di scaricare PhpMyAdmin da www.phpmyadmin.net, è un programma favoloso). Altrimenti potete utilizzare un altro ambiente, che si chiama **XAMPP** e va benissimo anche per chi usa un Mac: trovate l’ambiente di sviluppo all’indirizzo www.apachefriends.org.

Io preferisco WAMP (e lo conosco molto meglio), quindi faremo riferimento a questo ambiente, perché è molto semplice, è veloce da utilizzare e non richiede alcuna conoscenza di base! A questo punto facciamo un bel **download** (è pubblicato con licenza GNU GPL, quindi è libero e gratis!) e **installiamo la piattaforma** sulla nostra macchina (durante l’installazione dovremo indicare un **browser** che ci servirà quando faremo le prove: indicate pure il vostro browser preferito). Clicchiamo sull’icona di WAMP per farlo avviare: noteremo che nella barra delle applicazioni apparirà un’icona a forma di W racchiusa in un rettangolo (nelle versioni più datate, l’icona era invece una specie di semicerchio), che in fase di caricamento sarà dapprima rossa, poi gialla e infine diventerà tutta verde: solo quando l’icona sarà tutta verde potremo iniziare a lavorare.

Facciamo ora clic con il **tasto sinistro** del mouse sull’icona: vedrete un elenco di voci, e per spiegarle partiamo dal fondo. L’ultimissima, “**Metti online”**, ci serve se vogliamo mettere in rete il nostro server (magari stiamo chattando con un’amica o con un amico e vogliamo fargli vedere la nostra creazione! Quindi all’inizio, visto che non avremo niente da far vedere, non ci servirà). Le due successive, “**Ferma tutti i servizi”** e “**Riavvia tutti**

i servizi”, ci servono se vogliamo “spegnere” oppure “riavviare” il nostro server virtuale, e la terzultima “**Avvia tutti i servizi”**, sarà da utilizzare dopo che avremo, per un motivo o per l’altro, “spento” il nostro server (la dicitura “tutti i servizi” si riferisce alle componenti dell’ambiente di sviluppo, e cioè il server stesso, il linguaggio PHP e il database).

Troviamo poi “**MySQL”**, “**PHP”** e “**Apache”**: queste tre voci hanno sottomenù con le varie configurazioni. Lasceremo quelle di default, che ci vanno benissimo così come sono: andranno cambiate solo se utilizzeremo dei CMS e ci serviranno delle modifiche particolari.

La prossima voce è “**www directory”** ed è importantissima, perché è la cartella dentro alla quale inseriremo tutti i nostri progetti. Proviamo a farci clic con il tasto sinistro: per ora è vuota, fatta eccezione per la pagina **index.php** che è la pagina principale del server virtuale. Ne parleremo in modo più approfondito tra pochissimo.

Abbiamo poi il nostro **PhpMyAdmin**, il programma che serve per gestire database di tipo **MySQL**: quando lavoreremo sui database, noi utilizzeremo MySQL.

L’ultima voce che troviamo partendo dal basso è “**Localhost”**: proviamo a cliccarci sopra. Vedrete che si aprirà il vostro browser preferito (… o almeno quello che avete indicato al momento dell’installazione) con la pagina principale di Wampserver, e cioè quell’*index.php* di cui si diceva poco fa: la pagina ci mostra la configurazione del server, i “tools” (che sono i già citati programmi per il database assieme a **phpinfo()**, la pagina che ci mostrerà le configurazioni del PHP) e infine i nostri progetti! Per adesso la colonna è vuota, ma per riempirla si fa prestissimo.

Andiamo nella www directory, creiamo una cartella e chiamiamola “nuovosito”: torniamo sul Localhost e vedremo che nella colonna “Your Projects” troveremo “nuovosito”.

Come riempire questa cartella? Lo vedremo nel prossimo capitolo!

2. Impariamo i primi passi

Bene, dopo aver installato l'**ambiente di sviluppo** è ora di cominciare a **scrivere un po' di codice**! La prima cosa da sapere è che possiamo inserire **codice PHP** in **qualsiasi punto della pagina**: all'inizio, a metà, alla fine… dove vogliamo. Però prima di scrivere il codice… dobbiamo creare la pagina!

Apriamo quindi la cartella "www" di WAMP e creiamo una nuova cartella, a cui daremo il nome che più ci aggrada: per questo esempio potremmo chiamarla *PrimoProgetto*. Apriamo la cartella *PrimoProgetto* e creiamo una pagina che chiameremo *index.php*: esattamente come per i siti in HTML, la prima pagina deve chiamarsi "index". Cambia, ovviamente, l'estensione.

Bene, adesso creiamo la **struttura HTML** come siamo abituati: doctype, tag html, insomma tutto ciò che serve per creare una pagina HTML in accordo con gli **standard internazionali**, che **dovremo rispettare sempre** anche progettando un sito in PHP!

Creata la nostra pagina siamo finalmente **pronti per aprirla con il nostro editor preferito** (io consiglio di installare **Notepad++**) e per **scrivere il nostro primo codice**! Tutto il **testo in PHP** deve essere compreso tra un'**etichetta** (o **tag**, come preferite, io uso i due termini indistintamente) **di apertura e un'etichetta di chiusura**. Le etichette di apertura e chiusura sono queste:

```
<?php



?>
```

All'interno di queste due tag andrà inserito **tutto il nostro codice**, stando bene attenti a non commettere errori… vedremo tra pochissimo quanto sono facili le sviste con il PHP. Il primo **comando** che impariamo si chiama "echo". Questo comando è un **costrutto** che serve per **stampare a schermo una stringa di testo**, ovvero ogni volta che digiteremo il comando *echo*, seguito da una stringa tra parentesi e **apici o doppi apici** e **chiuso da un punto e virgola**, vedremo comparire (quando apriremo la pagina con il browser), quella scritta che abbiamo inserito nel comando. Per vedere il risultato con WAMP, vi basterà cliccare sull'icona nel menù delle applicazioni e cliccare quindi su "Localhost": si aprirà il vostro browser preferito con l'elenco dei progetti, nel nostro caso "PrimoProgetto". Cliccate su "PrimoProgetto" e vi si aprirà la pagina *index.php*.

Ma passiamo ora a un esempio pratico del comando *echo*:

```
<?php

echo (“Ciao, io mi chiamo Federico!”);

?>
```

Dal momento che *echo* **non è una funzione** (vedremo poi più avanti cosa sono le funzioni), possiamo anche omettere le parentesi:

```
<?php

echo “Ciao, io mi chiamo Federico!”;

?>
```

E possiamo cambiare i doppi apici con gli apici:

```
<?php

echo ‘Ciao, io mi chiamo Federico!’;

?>
```

Gli apici o i doppi apici **devono sempre essere due**! Una cosa del genere non può esistere perché **restituisce un errore**:

```
<?php
```

```
echo ‘Ho visto un’anatra’;

?>
```

Se proviamo a scrivere questa cosa sopra e poi proviamo ad aprire la pagina *index.php* scrivendo, nella barra degli indirizzi del nostro browser, *http://localhost/PrimoProgetto/index.php*, vedremo comparire una scritta del genere:

Parse error: parse error, expecting `’,” or `’;” in **[percorso]** on line **[numero riga]**

Significa che manca un apice, perché il server interpreta l’apostrofo dell’anatra di cui sopra come un apice (e in effetti, abbiamo usato il carattere dell’apice, e non quello dell’apostrofo, che è diverso, anche graficamente). Come fare quindi se ci serve un apice che faccia da apostrofo? Semplice: utilizziamo il **carattere di escape**, che è nient’altro che la **backslash**, subito prima dell’apice:

```
<?php

echo ‘Ho visto un'anatra’;

?>
```

Con il carattere di escape il server capirà che **non dovrà tener conto** di quell’apice. Ah, un’altra cosa a cui fare bene attenzione: **non confondete** gli apici con gli apostrofi e i doppi apici con le virgolette. Se scriveremo il codice con il blocco note o con un editor come Dreamweaver non avremo problemi, ma se avremo l’insana idea di scrivere il codice con, che so, Writer, i doppi apici vi verranno automaticamente trasformati in virgolette e se proverete a copiare il codice con le **virgolette al posto dei doppi apici** in una pagina PHP, non vi funzionerà niente. E altra cosa importante, **non dimenticate il punto e virgola alla fine**.

State bene attenti a non confondere apici e doppi apici tra di loro (per esempio non potete aprire un *echo* con un apice e chiuderlo con un doppio apice): sembrano banalità ma si possono trascorrere mezz’ore con complesse funzioni PHP che dànno errore per scoprire

alla fine, dopo aver controllato ogni singola riga di complicatissimi cicli o blocchi di istruzioni, che è tutto causato da un *echo* chiuso male e a cui magari non si è neppure fatto caso. Eh, per il PHP ci vuole **un po' di pazienza**.

Tornando a noi, possiamo utilizzare *echo* anche per **stampare tag HTML**:

```
<?php

echo '<strong>Ciao</strong>, io mi chiamo <em>Federico</em>!<br />Come va?';

?>
```

E possiamo **concatenare** più stringhe utilizzando, tra una stringa e l'altra (ognuna compresa tra apici o doppi apici), **il punto** e chiudendo il tutto, al solito, con il punto e virgola:

```
<?php

echo 'Questa'. ' è una'. ' concatenazione '. 'di stringhe!';

?>
```

In questo caso però **state attenti agli spazi** se non vorrete vedere le parole tutteattaccatecosì. La concatenazione ci sarà molto utile in futuro per fare alcune operazioni, soprattutto quando vedremo come far interagire **PHP e MySQL**: quindi dobbiamo impararla bene.

Infine impariamo a usare **i commenti**. Ci sono due modi per commentare in php: **su una riga o su più righe**. I commenti su una riga si aprono con un **doppio slash**:

```
<?php
```

```
// questo è un commento su una riga



?>
```

Mentre quelli **multiriga** si aprono con **backslash e asterisco** e si chiudono con **asterisco e slash**, proprio come i **commenti CSS**:

```
<?php



/* questo è un commento

multiriga */



?>
```

Non ci sono particolari raccomandazioni sull'uso dei commenti: dobbiamo solo **stare attenti a dove metterli**. Non possiamo per esempio metterli tra il comando *echo* e il doppio apice:

```
<?php



echo //questo è sbagliato e restituirà un errore "Ciao a tutti!";



?>
```

I commenti, come nel caso di CSS, ci serviranno se vorremo **descrivere alcune parti di codice** a chi le leggerà, per esempio. Oppure per **ricordarci che cosa fa una certa funzione**. Insomma, ognuno fa dei commenti l'uso che più ritiene necessario! C'è anche chi non li utilizza, ma io **consiglio sempre di usarli**, perché quando avremo a che fare con funzioni molto complesse ci sarà utile commentare alcune righe per ricordare meglio certi passaggi.

3. Semplifichiamoci la vita: il nostro primissimo sito in PHP!

Per noi che siamo abituati a creare siti in **HTML**, i **CSS** sono una **bellissima invenzione**: con pochi clic possiamo **cambiare grafica, colori, immagini di sfondo**, ampiezza dei *div* al nostro sito. Sono un po' come dei **vestiti di sartoria** che cuciamo su misura per il corpo della nostra **pagina web**.

Ma ci sono punti a cui anche i CSS non possono arrivare, purtroppo. Supponiamo di aver creato un sito di **cinquanta pagine**, con un **bel menù** come questo:

```
<div id="menu">

<ul>

<li>Home page</li>

<li>Chi siamo</li>

<li>Cosa facciamo</li>

<li>Servizi</li>

<li>Portfolio</li>

<li>Dove siamo</li>

<li>Contatti</li>

</ul>

</div>
```

Supponiamo anche che, per una qualsiasi ragione, dobbiamo **modificare questo menù**, magari **aggiungendo una voce**. Con HTML l'unico modo per **aggiornare il menù** è aprire tutte le nostre cinquanta pagine e in ognuna di esse aggiungere la voce mancante. Una bella seccatura, non c'è che dire.

A questo punto entra in gioco uno dei **costrutti più utili** di PHP: si chiama **include** e serve, come dice il nome stesso, per "includere" in una pagina **un file esterno**.

Ma vediamone subito l'utilizzo. Prima di tutto dobbiamo **creare un nuovo file** in cui inseriremo **soltanto il menù**: pertanto inseriremo nel nuovo file la *ul* e tutto il suo contenuto, ma non il *div*. Possiamo assegnare a questo nuovo file il suffisso *txt* o ancora

meglio il suffisso *html*. Chiameremo quindi il file “menu.html”.

A questo punto portiamoci nella pagina principale (per esempio, “index.php”) e, al posto della *ul*, **utilizziamo il costrutto include**, sempre aprendo il codice PHP come abbiamo imparato nel capitolo precedente:

```
<div id=“menu”>

<?php



include(“menu.html”);



?>

</div>
```

Se aprirete la pagina, noterete che avrà **un aspetto identico** a quello della pagina creata solo con il codice HTML, perché questo costrutto non fa altro che **inserire** all’interno della pagina **il contenuto di un file esterno**. Osservando il codice avrete già intuito **la sintassi del costrutto**: *include* con, tra parentesi, il nome del file racchiuso tra doppi apici (o apici), il tutto chiuso da punto e virgola.

In questo modo, ogni volta che dovremo modificare il menù, **sarà sufficiente** apportare le modifiche **al solo file** “menu.html” e automaticamente, se avremo utilizzato “include” in tutte le cinquanta pagine, le modifiche saranno estese a tutto il sito! Davvero una grande comodità!

È inteso che il nome del file che includiamo tra parentesi e apici o doppi apici deve trovarsi **nella stessa cartella del file principale**. Altrimenti **dovremo specificare il percorso** nello stesso modo in cui specifichiamo i percorsi, per esempio, di file html con l’etichetta *a*, ma senza poter utilizzare percorsi completi (per esempio: http://www.miosito.com/includes/menu.html)… i percorsi devono **sempre essere relativi**:

```
<?php



include(“includes/menu.html”);
```

```
?>
```

```
<?php

include("../menu.html");

?>
```

Include è del tutto simile a **echo**, quindi anche se omettiamo le parentesi non succede niente di strano. Sappiate però che è indifferente usare o meno le parentesi. È però molto importante **non sbagliare i percorsi**: nel caso in cui non dovessimo specificare il percorso corretto, comparirebbe un **warning** nella nostra pagina, e sinceramente non sarebbe bello da vedere. E ovviamente il file che vogliamo includere non sarà affatto incluso se sbagliamo il percorso!

Il costrutto *include*, come abbiamo visto, è anche sufficiente per creare **il nostro primissimo sito in PHP** e ci aiuta già a risolvere un grosso problema. A proposito: attenti a chiamare *include* "funzione" (cosa che avviene in **moltissimi siti**), perché **non è una funzione**. Lo so, vi starete ancora chiedendo cos'è una funzione. Tra non molto ci arriveremo!

Per completezza dobbiamo sapere che esiste anche **un altro costrutto per includere file esterni**, che si chiama **require** e funziona **allo stesso modo di include**:

```
<?php

require("menu.html");

?>
```

Qual è la **differenza tra include e require**, dal momento che il risultato prodotto è lo

stesso? La differenza sta nella **gestione degli errori**: come abbiamo visto, *include*, nel caso in cui ci sia qualcosa che non va (come il **percorso sbagliato**) restituisce un **warning**, ovvero una scritta che vi avverte dello sbaglio: questo però **non impedisce** che il **resto della pagina** si possa vedere correttamente.

Require invece restituisce un **fatal error**: se c'è qualcosa che non va nel *require*, **si blocca tutta la pagina**. Vi invito a fare qualche prova con la vostra pagina "index.php" per notare la differenza, anche perché il PHP si impara, più che con la teoria, con la pratica e con l'esercizio!

4. Le variabili, che cosa sono e a cosa servono

È arrivato il momento di **entrare più nello specifico**, parlando di **variabili**. Cosa sono le variabili? Sono **identificatori a cui sono associati certi valori** che, come suggerisce il termine "variabili", possono variare all'interno del codice. Così forse è un po' dura da capire, e facciamo un esempio stupido ma efficace.

Supponiamo di avere un bicchiere, una scatola e un piatto: questi sono **gli identificatori**. Supponiamo poi di avere una mela, una pera e una ciliegia. Questi sono **i valori**. Inseriamo la ciliegia nel bicchiere, la mela nella scatola e mettiamo la pera sul piatto: abbiamo **associato i valori agli identificatori**. Ovviamente potremo poi inserire la pera nella scatola e mettere la mela sul piatto: questo perché il valore associato agli identificatori può **variare** all'interno del codice. L'operazione con cui si dichiara **l'assegnamento** di un valore a un identificatore è detta **"inizializzazione"** (o **"dichiarazione"**).

Come si inizializzano le variabili in PHP? In questo modo:

```
<?php



$nome = "Federico";

$cognome = "Giannini";



?>
```

In questo esempio abbiamo **dichiarato due variabili**: una chiamata **$nome** e un'altra chiamata **$cognome**, e abbiamo assegnato i valori "Federico" alla prima e "Giannini" alla seconda. Avrete già intuito **la sintassi**: l'identificatore (cioè il nome della variabile) deve essere **sempre preceduto dal simbolo del dollaro** e tra quest'ultimo e l'identificatore **non deve esserci nessuno spazio**. E ovviamente il valore, in caso di **stringhe di testo** sarà racchiuso **tra apici o doppi apici** (sempre!). Ho specificato "in caso di stringhe di testo", perché per quanto riguarda **i numeri**, questi ultimi possono anche non essere racchiusi tra apici o doppi apici:

```
<?php
```

```
$numero = 3;



?>
```

Importante: non possiamo dare alle variabili il primo nome che ci passa per la testa perché ci sono **alcune regole da rispettare**. Regola numero uno: **il primo carattere** dell'identificatore **non può essere un numero**. Quindi se abbiamo intenzione di chiamare una variabile $1a, dobbiamo sapere che il programma restituirà un errore. Come **primo carattere** possiamo utilizzare soltanto **una lettera** o al più **un underscore**. Regola numero due: **i nomi** delle variabili **sono case sensitive**. Quindi $nome e $Nome **non sono la stessa cosa**. Regola numero tre (o meglio, consiglio più che regola): utilizzate solo lettere, numeri o al massimo underscore nei vostri identificatori. Alcuni caratteri "speciali" sono ammessi, ma la maggior parte (giusto per fare qualche esempio: virgola, asterisco, parentesi) non può far parte dell'identificatore di una variabile.

Abbiamo quindi tutti gli elementi sufficienti per **dichiarare una variabile**. Ma **a cosa servono** le variabili? Ci servono per **memorizzare informazioni** che dovremo utilizzare nel codice. Supponiamo di creare una pagina in cui dobbiamo inserire una scritta "Per maggiori informazioni contattarci ai nostri indirizzi e-mail".

Potremmo dichiarare una variabile in questo modo:

```
<?php



$stringa = "Per maggiori informazioni contattarci ai nostri indirizzi e-mail";



?>
```

E, nei punti in cui dobbiamo far comparire la scritta, inseriremo:

```
<?php



echo $stringa;
```

```
?>
```

E, richiamando tramite il costrutto *echo* la variabile **$stringa**, comparirà sulla nostra pagina la scritta **così come l'abbiamo definita** nella dichiarazione della variabile $stringa. Attenzione però! Se creiamo una pagina "index.php" e una pagina "contatti.php", **non possiamo richiamare** in "contatti.php" una variabile dichiarata in "index.php" e viceversa! Le variabili possono essere **richiamate soltanto nella pagina in cui sono state dichiarate**, mi raccomando! In caso contrario ci sarà un errore.

Come fare allora per far stampare la stringa in tutte le pagine del sito? Alcuni forse avranno già intuito: usiamo il costrutto *include*. Creiamo quindi una pagina "scritta_da_richiamare.php" e inseriamo all'interno **soltanto la dichiarazione** della variabile. In tutte le altre pagine non dovremo far altro che includere il file "scritta_da_richiamare.php" e quindi richiamare la variabile con il costrutto echo, in questo modo:

```
<?php

include ("scritta_da_richiamare.php");

echo $stringa;

?>
```

Possiamo anche **concatenare stringhe e variabili**, utilizzando lo stesso metodo che abbiamo visto nel capitolo precedente:

```
<?php

$inizio = "Questo è l'inizio";

echo $inizio. " di questa frase.";
```

```
?>
```

Sulla pagina comparirà la scritta “Questo è l’inizio di questa frase”. Possiamo fare la stessa cosa **con i numeri**:

```
<?php

$numero_di_mele = “3”;

echo “Ci sono “. $numero_di_mele. ” mele sul tavolo.”;

?>
```

Comparirà sulla pagina la scritta “Ci sono 3 mele sul tavolo”. A questo punto, visto che abbiamo preso **un po’ di confidenza** con le variabili, è il momento di sapere qual è la **differenza tra apici e doppi apici**, perché sì, c’è una differenza nell’utilizzo di questi due caratteri per includere stringhe di testo.

Riproviamo con l’esempio precedente, aggiungendo un *echo* con i doppi apici e un *echo* con gli apici;

```
<?php

$numero_di_mele = “3”;

echo “Ci sono $numero_di_mele mele sul tavolo.<br />”;

echo ‘Ci sono $numero_di_mele mele sul tavolo.’;

?>
```

Nel primo caso la scrittà stampata sarà questa: “Ci sono 3 mele sul tavolo”. Ma nel

secondo caso sarà questa: “Ci sono $numero_di_mele mele sul tavolo”. La differenza tra apici e doppi apici sta quindi nella **gestione delle variabili**: i **doppi apici interpretano le variabili** e inseriscono **correttamente** il loro valore in una stringa di testo, **gli apici invece no**. Quindi se dobbiamo **includere variabili in un testo** con il costrutto *echo* è consigliabile utilizzare **i doppi apici** per **evitare inconvenienti** come quello descritto sopra.

Infine, ultimo argomento per questo capitolo: come **cambiare il valore di una variabile**. Basta semplicemente inserire una nuova dichiarazione, in questo modo:

```
<?php

$numero_di_mele = “3”;

echo “Ci sono $numero_di_mele mele sul tavolo.<br />”;

$numero_di_mele = “6”;

echo “Adesso ce ne sono $numero_di_mele.”;

?>
```

Il risultato sarà questo:

Ci sono 3 mele sul tavolo.

Adesso ce ne sono 6.

5. Proviamo qualche operazione con le variabili

Nel capitolo precedente abbiamo affrontato una **panoramica** piuttosto ampia sulle **variabili**. Dico “piuttosto” perché non abbiamo invece parlato di un **aspetto importante**: con le variabili **si possono eseguire operazioni**! E con “operazioni” intendiamo proprio **operazioni matematiche**! Abbiamo visto che il **valore associato a una variabile** può essere un **numero**: è quindi del tutto lecito supporre che con questi “numeri” ognuno possa fare quello che vuole.

Le prime operazioni che ci hanno insegnato alle **scuole elementari** sono quattro: **addizione**, **sottrazione**, **moltiplicazione**, **divisione**. Il PHP ci fornisce alcuni **operatori** molto semplici e molto intuitivi per poter fare proprio queste operazioni. Poniamo di avere una variabile $x che è uguale a 6 e una variabile $y che è uguale a 3:

```
<?php



$x = 6;

$y = 3; // inizializziamo le variabili



$addizione = $x + $y;
// si utilizza semplicemente il carattere +

$sottrazione = $x - $y;
// anche qui molto facile, per la sottrazione si usa -

$moltiplicazione = $x * $y;
// per la moltiplicazione il carattere è l’asterisco *

$divisione = $x / $y;
// per la divisione invece si usa lo slash /



/* Se vogliamo fare una cosa carina, stampiamo le variabili e i risultati: */



echo “Primo numero: $x<br />”;

/* mi raccomando! Se inserite la variabile direttamente nel testo

USATE I DOPPI APICI come abbiamo visto nel capitolo precedente */
```

```
echo ‘Secondo numero: ‘. $y. ‘<br /><br />’;

// altrimenti usate la concatenazione…


echo “Risultato addizione: $addizione<br />”;

echo “Risultato sottrazione: $sottrazione<br />”;

echo “Risultato moltiplicazione: $moltiplicazione<br />”;

echo “Risultato divisione: $divisione<br />”;


?>
```

Se avremo fatto tutto bene, otterremo questo:

Primo numero: 6

Secondo numero: 3

Risultato addizione: 9

Risultato sottrazione: 3

Risultato moltiplicazione: 18

Risultato divisione: 2

Esiste anche **un’altra operazione** in PHP, che si chiama **modulo**. Ricordate quando alle scuole elementari la maestra vi faceva fare **divisioni** tra due numeri e il più grande **non era multiplo** del più piccolo? Per esempio, 9 diviso 5: adesso diremmo che il risultato è 1,8 ma all’epoca dicevamo che il risultato è “**1 con il resto di 4**”. Ecco il modulo è proprio questo: restituisce il **resto della divisione**, e il carattere che si usa è il percento (%). Per cui:

```
<?php


$x = 9;
```

```
$y = 5;



$modulo = $x % $y;



echo “Dividendo $x per $y, il resto è $modulo”;



?>
```

Otterremo la frase “Dividendo 9 per 5, il resto è 4”. Ovviamente nessuno vi vieta di velocizzare le operazioni in questo modo:

```
<?php



$x = 9;

$y = 5;



echo “Dividendo $x per $y, il resto è “. $x % $y;

// abbiamo usato la concatenazione e abbiamo risparmiato una variabile



?>
```

Esiste poi un altro tipo di operazioni, e cioè le **operazioni di calcolo**, che servono per **aumentare**, **diminuire**, **moltiplicare** o **dividere il valore della variabile**. Per eseguire le operazioni di calcolo, basta soltanto utilizzare gli operatori che abbiamo appena visto e farli **seguire immediatamente** dal segno di **uguale** (=):

```
<?php



$x = 6;
```

```
$x += 3; //il valore di $x viene aumentato di 3

echo “$x<br />”;



$x = 6;



$x -= 3; // il valore di $x viene diminuito di 3

echo “$x<br />”;



$x = 6;



$x *= 3; // il valore di $x viene moltiplicato per 3

echo “$x<br />”;



$x = 6;



$x /= 3; // il valore di $x viene diviso per 3

echo “$x<br />”;

?>
```

Se avete scritto tutto bene, otterrete in colonna i numeri 9, 3, 18 e 2. Dopo ogni operazione abbiamo riscritto sempre “$x = 6;” perché altrimenti l’operazione **viene svolta sull’ultimo valore associato** alla variabile $x. Se scriviamo per esempio questo:

```
<?php



$x = 6;



$x += 3; //il valore di $x viene aumentato di 3
```

```
echo “$x<br />”;



$x -= 3; // il valore di $x DIVENTATO 9 viene diminuito di 3

echo “$x<br />”;



?>
```

Invece di 9 e 3 i risultati saranno 9 e 6, perché la sottrazione viene eseguita **sull’ultimo valore associato a $x**, che in seguito all’addizione è diventato 9. Quindi **attenzione** quando si fanno queste operazioni.

Infine, due operazioni molto importanti: l’**incremento** e il **decremento**, che saranno utilissime quando affronteremo i cicli. Si utilizzano per **aumentare o diminuire di una unità** il valore di una variabile e per utilizzarli basta scrivere la variabile **seguita da due caratteri +** per l’incremento e **due caratteri –** per il decremento (ovviamente il tutto seguito da punto e virgola, non dimentichiamolo mai):

```
<?php



$x = 6;

$x++; // incremento di una unità: $x diventa 7

echo “$x<br />”;



$x = 6;

$x—; // decremento di una unità: $x diventa 5

echo “$x<br />”;



?>
```

Se avremo fatto tutto bene, avremo ottenuto stampati in colonna sul nostro schermo i numeri 7 e 5. Anche in questo caso vale la raccomandazione fatta poco fa: se vogliamo

riportare $x al suo valore originario **dovremo di nuovo** dichiarare 6 come valore. Se avessimo omesso questo passaggio nel codice sopra, avremmo ottenuto “6” in seguito al decremento, perché quest’ultimo sarebbe stato eseguito con la $x avente 7 per valore.

6. Cenni sui tipi

In PHP ci sono diversi **tipi** di variabili. Il nome **“tipo”** non è usato a caso, perché in informatica indica un qualcosa **di ben preciso**: è l’**insieme di valori** che può assumere una **variabile**. Finora ne abbiamo visti due: gli **interi** e le **stringhe**. Ma oltre a questi ce ne sono tanti altri.

I “tipi”, detti anche **“tipi di dato”**, possono essere classificati in **primitivi** o **composti**. Appartengono alla prima categoria **gli interi, le stringhe, i numeri in virgola mobile, i booleani**. Appartengono invece alla seconda categoria **gli array e gli oggetti**. In questo ebook non ci occuperemo degli oggetti.

Vediamo quindi i tipi con cui non abbiamo ancora fatto la conoscenza. Tralasciamo per adesso gli **array**, ai quali dedicheremo un capitolo intero del nostra ebook (il prossimo!), e concentriamoci sui **numeri in virgola mobile** e sui **booleani**. I numeri in virgola mobile non sono altro che valori che rappresentano **numeri decimali**, e possono essere compresi tra un intervallo che va da 1.7E-308 a 1.7E+308, quindi si parla di numeri davvero alti. Possono essere espressi sia con la **virgola** (o meglio… con il **punto**), come siamo abituati, sia in **notazione scientifica**:

```
<?php

$a = 0.03;

$b = 3.0E-3;

?>
```

Abbiamo visto i numeri in virgola mobile un po’ per completezza, perché in genere per creare siti web si usano poco. Vediamo quindi i **booleani**, con i quali invece dovrete avere molta confidenza in futuro! I booleani (**“variabili booleane”**) sono variabili che possono assumere soltanto **due valori**: **vero o falso**. Prendono il nome da un matematico nonché filosofo inglese, **George Boole** (1815 – 1864), noto per essere stato il padre della logica matematica booleana (ovvero la base della logica dei computer moderni).

Si possono scrivere in questo modo:

```
<?php

$a = true;
```

```
$b = false;

?>
```

I booleani ci servono quindi per assegnare **valori di verità** alle nostre variabili, o anche per controllare il risultato di certe operazioni o certi confronti: tutto questo ci tornerà davvero utile quando vedremo, tra pochi capitoli, i blocchi condizionali e i cicli.

Cosa significa che i booleani servono per controllare i risultati di certe operazioni? Significa che potremo verificare se il risultato di un'operazione **è vero o è falso**, sempre attraverso un blocco condizionale o un ciclo. Supponiamo, per esempio, di dover sapere se il risultato di una operazione $a è uguale a quello di una operazione $b: se è uguale, stamperemo la scritta "uguale" e se è diverso stamperemo la scritta "diverso".

```
<?php

$x = 2;

$y = 3;

$a = $x + $y;

$z = 1;

$t = 4;

$b = $z + $t;

/*

A questo punto con un blocco condizionale

controlleremo se il risultato di "a"

è uguale al risultato di "b"

e in caso affermativo faremo stampare "uguale"

al programma… in caso contrario

faremo stampare "diverso".

In questo caso, il risultato del confronto

$a == $b è "true" (con due uguali).

Se $x fosse stato 3, il risultato del confronto
```

```
sarebbe stato “false”.

*/

?>
```

Ovviamente i confronti si possono fare in diversi modi… adesso abbiamo visto **quello più semplice**, $a == $b, per verificare se due variabili sono uguali (si scrivono due uguali perché i due uguali sono un operatore di confronto, mentre il singolo uguale è un operatore di assegnazione: serve, come sappiamo e come abbiamo visto anche nell’esempio precedente, per assegnare il valore a una variabile. I due uguali invece servono per confrontare due variabili diverse). In caso **affermativo**, il risultato del confronto sarà **true** e in caso **contrario false**.

Ma ci sono altri operatori, e ora li vediamo uno per uno:

```
<?php

$a == $b
/* per controllare se sono uguali: in caso affermativo il risultato sarà true */

$a != $b
/* per verificare se $a è diverso da $b: in caso affermativo il risultato sarà true */

$a > $b
// per verificare se $a è maggiore di $b

$a < $b
// per verificare se $a è minore di $b

$a >= $b
// per verificare se $a è maggiore-uguale a $b

$a <= $b
// per verificare se $a è minore-uguale a $b

?>
```

I confronti non possono essere fatti solo con i **numeri**, ma si possono fare anche **con le stringhe**! In questo caso le prime lettere dell’alfabeto saranno **minori rispetto alle ultime** (la a è minore della b, la c è minore della f, la s è maggiore della m e così via). C’è poi da sapere che le lettere **minuscole sono maggiori delle maiuscole** e tutte le lettere **sono maggiori dei numeri**. Esempio:

```
<?php



$a = “Ciao”;

$b = “a tutti”;

$c = “i lettori!”;

$d = “10 volte”;

$a > $b;
/* risultato falso, perché “a tutti” è maggiore di “Ciao” in quanto inizia per minuscola */

$c > $b; // vero, perché la “i” è maggiore della “a”

$b > $d; // vero, perché le lettere sono maggiori dei numeri

?>
```

Esistono poi **gli operatori logici**, ovvero operatori che servono per **combinare valori booleani**, quindi per combinare i risultati di più operazioni. Supponiamo di avere quattro variabili **di tipo intero**: $a, $b, $c e $d. Supponiamo di voler stampare la scritta “tutto vero” se $a è maggiore di $b e $c è maggiore di $d. Dovremo comportarci così:

```
<?php

$a = 3;

$b = 2;

$c = 6;

$d = 5;

// il controllo dell’esempio si utilizza con l’operatore && (oppure AND)

$a > $b && $c > $d; // risultato true

$a > $b AND $c > $d; // equivalente all’espressione sopra

?>
```

Ci sono altri operatori logici:

```
<?php

$a = 3;

$b = 2;

$c = 6;

$d = 5;

$a > $b || $c < $d;

/* controlla se $a è maggiore di $b OPPURE se $c è minore di $d: nel caso in cui

una delle due condizioni sia vera, il risultato sarà vero. Si può scrivere anche così: /*

$a > $b OR $c < $d;



$a > $b Xor $c < $d;

/* Xor si chiama “or esclusivo”, e controlla che il primo risultato sia vero e il secondo sia falso:

in questo caso il risultato sarà true /*

?>
```

Infine merita un ultimo cenno il **casting**, che non ha niente a che vedere con quello del mondo dello spettacolo: in PHP (e in programmazione) il casting è un’operazione che serve per **“convertire”** una variabile **da un tipo all’altro**. Non abbiamo detto che, quando si dichiara una variabile, se ne può anche specificare il tipo, e lo si fa **indicandolo tra parentesi tonde prima del nome** della variabile:

```
<?php

(int) $a = 3;

(string) $b = ‘Ciao’;

(float) $c = 0.03; // numeri in virgola mobile

(boolean) $a = true;

?>
```

Per effettuare il casting, ovvero la conversione, basta semplicemente specificare il tipo, sempre tra parentesi tonde, prima del valore associato alla variabile:

```
<?php

$a = (string) 3;

?>
```

7. Gli array

Dobbiamo pensare a un **array** (termine che si può tradurre in italiano con la parola **“vettore”**, ma il 99% delle volte sentirete chiamare questo **tipo** con il suo nome in inglese) come a un **contenitore** che ha **una serie di elementi** e **ognuno di essi** ha **un proprio indice**.

La questione posta in questi termini forse può risultare un po’ ostica, quindi facciamo un esempio concreto. Possiamo associare l’idea di **array** a quella del **registro per l’appello**, quando andavamo a **scuola**: il registro è **l’array**, i nomi nostri e dei nostri compagni di classe sono **gli elementi**, il numero che ci contraddistingueva (soprattutto quando i professori estraevano a sorte per interrogare… !) è **l’indice**. L’unica differenza tra registro è array consiste nel fatto che **gli indici** nel registro partono da 1, invece nell’array **partono da 0**. L’array è **molto utile** perché è una **collezione di dati ordinata**, e quando vedremo come utilizzare in modo combinato **PHP** e **MySQL** faremo **largo uso di array**.

Nei capitoli precedenti abbiamo visto come **si inizializza una variabile**, e per quanto riguarda l’array il modo in cui viene definito è **un pochino diverso** proprio per il fatto che a un array **non si associa un solo valore** (come a una variabile), ma si possono associare **tanti valori**. Ci sono **due modi** per dichiarare un array, vediamoli entrambi:

```
<?php



$array[] = “Federico”;

$array[] = “Ilaria”;
$array[] = “Gabriele”;



// questo è il primo modo



$array = array(“Federico”, “Ilaria”, “Gabriele”);



// questo è il secondo modo



?>
```

Quindi, avrete sicuramente intuito che per far capire al **preprocessore** che stiamo **dichiarando un valore di un array**, dobbiamo far seguire al **nome dell'array** le **parentesi quadre vuote**. In questo modo l'array **viene riempito**, e gli indici **vengono assegnati**, a partire **da zero**, agli elementi **nell'ordine in cui li abbiamo dichiarati** (Federico avrà indice 0, Ilaria 1, Gabriele 2, ed eventuali elementi successivi avranno indice 3,4,5… e così via). Stessa cosa vale per il secondo modo: solito simbolo del dollaro seguito dal nome dell'array, e dopo l'uguale **non dimentichiamo il comando "array" seguito dagli elementi** (che devono trovarsi **tra parentesi tonde** -tutti- e **tra doppi apici** -ognuno- separati l'uno dall'altro **da virgole**).

Possiamo anche **dichiarare esplicitamente** l'indice. Supponiamo per esempio di voler inserire gli elementi in ordine alfabetico dichiarandoli però nello stesso ordine dell'esempio precedente. Non dovremo far altro che indicare l'indice tra le parentesi quadre:

```
<?php



$array[0] = "Federico";

$array[2] = "Gabriele";

$array[1] = "Ilaria";



?>
```

Può capitare di scrivere una cosa del genere:

```
<?php



$array[1] = "Federico";

$array[2] = "Ilaria";

$array[3] = "Gabriele";



?>
```

In questo caso **non ci saranno errori**, ma il primo elemento dell'array (il numero zero insomma), ovviamente, **non sarà definito**, quindi quando cercheremo di ricavarne il valore, magari con una stampa tramite il comando *echo*, questo sarà il risultato che vedremo sul sito:

Notice: Undefined offset: 0 in **C:\wamp\www\PrimoProgetto\index.php** on line **29**

Dove "29" è il numero della riga in cui compare il nostro comando echo. A proposito: come facciamo per **stampare a schermo** gli elementi del nostro array? È molto semplice:

```
<?php



$array[] = "Federico";

$array[] = "Ilaria";

$array[] = "Gabriele";



echo $array[0] . "<br />"; // stampiamo l'elemento 0, Federico

echo $array[1] . "<br />"; // stampiamo l'elemento 1, Ilaria

echo $array[2] . "<br />"; // stampiamo l'elemento 2, Gabriele



?>
```

In questo modo otteniamo gli elementi del nostro array tutti uno sotto l'altro, grazie al <br /> che **concateniamo** alla stampa dell'elemento. Quando ci occuperemo dei controlli iterativi vedremo che ci sarà anche **un modo più rapido** per stampare gli elementi, perché finché abbiamo un array con quattro elementi possiamo anche utilizzare il codice di esempio qua sopra, ma quando avremo a che fare con array di decine di elementi (e soprattutto se non sapremo **quali valori** corrispondono a **quali indici**), questo modo di stampare l'array sarà inservibile.

Se ci venisse in mente, per un qualche bizzarro motivo, di scrivere semplicemente "echo $array", il risultato che otterremo sarà semplicemente la scritta "Array" sul nostro schermo.

Nel caso in cui invece scegliessimo di conferire **lo stesso indice a due elementi**, come nell'esempio seguente:

```
<?php



$array[0] = "Federico";

$array[1] = "Ilaria";

$array[1] = "Gabriele";



?>
```

Automaticamente l'elemento "Ilaria" verrebbe **eliminato dall'array** e al suo posto verrebbe inserito l'elemento "Gabriele". Possiamo anche assegnare l'indice **ai primi due elementi** e non assegnarlo al terzo, e in questo caso l'elemento con l'indice non dichiarato assumerà **il primo indice disponibile**:

```
<?php



$array[0] = "Federico";

$array[1] = "Ilaria";

$array[] = "Gabriele";

// l'elemento "Gabriele" assumerà il primo indice disponibile, e cioè 2



?>
```

E infine possiamo anche aggiungere un elemento a un array in un secondo momento, magari dopo aver fatto una stampa:

```
<?php
```

```
$array[0] = “Federico”;

$array[1] = “Ilaria”;



echo $array[0] . “<br />”;

echo $array[1] . “<br />”;



$array[] = “Gabriele”;



echo $array[2] . “<br />”;



?>
```

Anche in questo caso l’elemento “Gabriele” assumerà **il primo indice disponibile** (ma avrete ovviamente intuito che possiamo anche specificarlo). Torniamo un attimo indietro, al secondo metodo che abbiamo utilizzato per dichiarare l’array. Se volessimo **farlo partire da 1** invece che da 0, dovremmo scrivere l’istruzione in questo modo:

```
<?php



$array = array(1 => “Federico”, “Ilaria”, “Gabriele”);



?>
```

La prima cosa dopo la parentesi tonda dovrà essere l’indice desiderato (nel nostro caso 1), seguito da **una specie di “freccia”** costruita con il **simbolo di uguale e la parentesi uncinata chiusa**, e quindi gli elementi ognuno **tra doppi apici** e separato dal successivo attraverso **una virgola**.

Abbiamo visto finora un array fatto **di sole stringhe**, ma possiamo anche creare **array di numeri**, per esempio:

```
<?php

$array_di_numeri = array(1, 2, 3);

?>
```

O anche array misti come questo:

```
<?php

$array = array(“Federico”, 1, 2);

?>
```

Possiamo anche inizializzare un **array vuoto**, che magari riempiremo **successivamente**:

```
<?php



$array = array();



?>
```

Infine diamo un’occhiata ai cosiddetti **“array associativi”**, ovvero array in cui gli indici **non sono numeri** come abbiamo visto finora **ma sono stringhe**, come nell’esempio seguente:

```
<?php



$array[‘primo_elemento’] = “Federico”;

$array[‘secondo_elemento’] = “Ilaria”;

$array[‘terzo_elemento’] = “Gabriele”;

// questo è un array associativo
```

```
echo $array[‘primo_elemento’] . “<br />”;

echo $array[‘secondo_elemento’] . “<br />”;

echo $array[‘terzo_elemento’] . “<br />”;

// questo è il modo in cui se ne stampano gli elementi



?>
```

Potete liberamente scegliere se includere l’indice **tra apici o doppi apici** ma occhio a dimenticarvi l’**inclusione**, perché una dichiarazione tipo $array[primo_elemento] può andare bene e l’elemento sarà aggiunto all’array ma vi comparirà una **notice**, che non è proprio un errore ma è comunque una notifica che vi fa capire che **c’è qualcosa che non va** e, a meno che non le disabilitiamo (cosa sconsigliatissima), non sono belle da vedere. Ecco cosa succede con una dichiarazione come “$array[php]”:

Notice: Use of undefined constant primo_elemento -

assumed ‘primo_elemento’ in **C:\wamp\www\PrimoProgetto\index.php** on line **26**

Infine, sempre per quanto riguarda gli **array associativi**, dobbiamo sapere che se vogliamo includere **la stampa** di un elemento dell’array all’interno di una frase stampata con *echo*, possiamo sì utilizzare la concatenazione come abbiamo fatto finora, ma possiamo anche **includere** l’elemento dell’array **direttamente nella frase** (che ovviamente, come ricorderete, dovrà essere compresa esclusivamente tra doppi apici), ma **con un’accortezza**, e cioè dovremo inserire l’elemento **tra parentesi graffe**:

```
<?php



$array[‘primo_elemento’] = “Federico”;

$array[‘secondo_elemento’] = “Ilaria”;

$array[‘terzo_elemento’] = “Gabriele”;



echo “Il primo elemento è {$array[‘primo_elemento’]}.”;



?>
```

Se dovessimo dimenticarci **le parentesi graffe**, ci sarà un errore. Con gli array si possono fare tante cose e si possono eseguire tante operazioni: ma di questo parleremo più avanti!

8. Le variabili “relative al web”

In PHP esistono alcune **variabili speciali**, che servono per gestire dati ricevuti da una **richiesta HTTP** (detto in modo più chiaro: servono per **generare contenuti** in base a **particolari richieste degli utenti**… per esempio **dopo la compilazione di un form**). **Rasmus Lerdorf**, nel suo libro *PHP*, chiama queste speciali variabili “**variabili relative al web**”.

Queste variabili sono dette anche “**variabili superglobali**”, perché sono disponibili **sempre**, in qualunque nostra pagina, anche **senza che noi le dichiariamo** (proprio perché sono **già definite** all’interno dello stesso linguaggio PHP). Si chiamano “**superglobali**” per distinguerle dalle variabili “**globali**”, che vedremo invece quando ci occuperemo delle funzioni.

Ma torniamo alle nostre variabili “relative al web”. L’esempio classico che si fa per comprenderle è quello del **form**. Un form non è altro che un **modulo** che serve per passare dati: per esempio, vogliamo far scrivere all’utente del nostro sito il suo nome in modo da fargli avere un messaggio di benvenuto personalizzato. Il codice del nostro form potrebbe quindi essere più o meno questo:

```
<form action=“benvenuto.php” method=“get”>

Inserisci il tuo nome:<br />

<input type=“text” name=“nome” id=“nome” /><br /><br />

<input type=“submit” value=“Invia!” />

</form>
```

Questo è tutto **HTML** ovviamente. Ma che cosa significa? Innanzitutto abbiamo **due input**, uno di tipo **“text”** per far immettere il nome al nostro utente, e uno di tipo **“submit”**, ovvero un bottone per permettergli di **inviare i dati** inseriti nel form. Ma la *tag* più interessante è **“form”** perché ha **due attributi**: uno si chiama **“action”** e deve contenere **la pagina verso la quale** l’utente sarà **reindirizzato** dopo aver cliccato sul bottone “Invia”, e l’altro invece si chiama **“method”** e può assumere **due valori**: **“get”** oppure **“post”**.

Nel caso dell’esempio qui sopra abbiamo scelto il metodo **get**: significa che i dati che inseriamo vengono **passati direttamente** allo URL della pagina, che sarà seguito **da un punto interrogativo** a sua volta seguito **dal nome del campo “input” e dai dati inseriti**.

Tutto ciò che viene dopo il punto interrogativo viene definito **“query string”**. Se per esempio inserisco il nome “Federico” nel campo del form precedente, verrò reindirizzato a una pagina che avrà come URL http://localhost/PrimoProgetto/benvenuto.php?nome=Federico. Semplice no?

A questo punto entrano in gioco le nostre **variabili relative al web**. Ora, noi vogliamo far stampare sullo schermo, quando l’utente viene reindirizzato, la scritta “Benvenuto [nome]!”, quindi nel nostro caso “Benvenuto Federico!”. Basterà utilizzare la variabile chiamata **$_GET** (tutte le variabili superglobali hanno i nomi **completamente in maiuscolo**). La variabile $_GET non è altro che **un array** a cui viene aggiunto **un elemento** che ha come indice “nome” e come valore “Federico” (si tratta quindi di **un array associativo**!).

Richiamare la variabile all’interno del codice e far stampare la scritta di benvenuto è semplicissimo, perché ci comportiamo **come con un normalissimo array associativo**, utilizzando, ovviamente, la **concatenazione**:

```
<?php

echo “Benvenuto “. $_GET[‘nome’] .”!”;

?>
```

Questo codice, ovviamente, non andrà nella pagina “index.php”, ma nella pagina “benvenuto.php”. Quindi, quando avremo inserito il nostro nome (nel mio caso Federico) e avremo cliccato sul pulsante di tipo submit, ci apparirà una pagina con la **scritta di benvenuto**, nel mio caso “Benvenuto Federico!”.

Se abbiamo **più di un campo** attraverso il quale far passare dati (per esempio invece della scritta con soltanto il nome vogliamo far stampare la scritta con nome e cognome), basterà aggiungere un input di tipo “text” al nostro form e i dati saranno separati nello URL da una **“e commerciale”**, quindi l’indirizzo, nel mio caso (“Federico Giannini”), assumerà questa forma: http://localhost/PrimoProgetto/benvenuto.php?nome=Federico&cognome=Giannini. Dal secondo elemento in poi, tutti i dati saranno separati sempre dalla nostra **ampersand** (altro nome della “e commerciale”).

Abbiamo poi un altro metodo, che si chiama **“post”**, e la differenza rispetto al metodo “get” consiste nel fatto che, con il metodo post, al nostro URL **non viene appeso alcunché**, ma i dati vengono **passati attraverso il server**, quindi la pagina a cui verremo reindirizzati sarà semplicemente la pagina “benvenuto.php” senza nessun dato attaccato con punto interrogativo (detta in modo più semplice, **get genera una query string** e **post**

invece no).

Quindi, per utilizzare un form con il metodo post, basterà semplicemente **cambiare l'attributo "method"** del nostro form in questo modo:

```
<form action="benvenuto.php" method="post">
```

E, ovviamente, utilizzare l'array associativo superglobale **$_POST** al posto di $_GET, in questo modo:

```
<?php


echo "Benvenuto ". $_POST['nome'] ."!";


?>
```

Tutto veramente molto facile anche se, naturalmente, bisogna fare un po' di esercizio! Quindi vi esorto fin da ora a provare questi codici, magari per **dare il benvenuto** agli utenti dei vostri siti con delle frasi personalizzate, può essere una cosa carina!

Per finire, una piccola nota storica: i nostry array associativi $_GET e $_POST sono stati introdotti soltanto a partire **dalla versione 4** di PHP: prima, per compiere le stesse operazioni, era necessario utilizzare le variabili $HTTP_GET_VARS e $HTTP_POST_VARS… beh, senza dubbio le nuove variabili superglobali sono molto più comode!

Ma prima di terminare il capitolo, voglio provare a rendere questo corso un po' meno passivo, quindi per la prima volta **vi propongo un esercizio**, per verificare l'apprendimento delle variabili relative al web.

Un hotel vi ha commissionato un sito all'interno del quale dovrà esserci un form per ricapitolare la prenotazione del cliente. Il cliente dovrà compilare il modulo inserendo nome e cognome, numero di notti di soggiorno, data di arrivo, scelta della colazione. Costruite un form che abbia queste caratteristiche e costruite la relativa pagina che, ricevendo i dati dal form, ricapitola la prenotazione al cliente. Potete usare get o post a

vostra scelta. Per quanto riguarda la colazione, dovrete usare dei “radio button” nel form. La soluzione dell’esercizio si trova nelle ultime pagine del libro, nell’apposito capitolo.

9. I blocchi condizionali if-else

Con questo capitolo iniziamo a entrare in temi un pochino più impegnativi rispetto a quelli che abbiamo visto finora. Qui vedremo una delle **strutture** più utili di **PHP**: i **blocchi condizionali**. Che cosa sono e a che cosa servono? Sono dei blocchi di **“istruzioni”**, e tra poco vedremo come funzionano, che **“osservano”** un **evento** scaturito da **un’azione dell’utente** e **si comportano di conseguenza** secondo **uno schema che avremo prefissato**. In parole povere, i blocchi condizionali servono per far sì che il nostro sito si comporti in un certo modo **se si verificano alcune circostanze**.

Facciamo un esempio molto semplice e molto carino. Vogliamo creare, sul nostro sito, **un form** con il quale **dare il benvenuto** ai nostri utenti, che dovranno inserire non soltanto il loro nome, ma anche il sesso: se il nostro utente sarà un ragazzo (o un signore) lo saluteremo con una scritta blu, se sarà una ragazza (o signora o signorina) la saluteremo invece con una scritta rosa.

Per fare questo abbiamo bisogno di alcuni strumenti: **il form** e le **variabili relative al web**, come abbiamo visto nel precedente capitolo e, appunto, il nostro **blocco condizionale if-else**.

Cominciamo a costruire il form, e per far immettere il sesso al nostro utente utilizziamo dei **radio button**:

```
<form action=“benvenuto.php” method=“post”>

Come ti chiami?<br />

<input type=“text” name=“nome” /><br /><br />

Ragazza o ragazzo?<br />

F <input type=“radio” name=“sesso” value=“F” /><br />

M <input type=“radio” name=“sesso” value=“M” /><br /><br />

<input type=“submit” name=“invia” value=“Entra nel sito!” />

</form>
```

Per l’esempio abbiamo scelto il metodo **“post”**, e come vedete è un form molto semplice. Il bello viene adesso, quando faremo stampare la scritta colorata a seconda del sesso dell’utente. Creiamo quindi la pagina “benvenuto.php” e facciamolo in questo modo

(adesso osserviamo il codice e subito dopo lo spieghiamo):

```
<?php

if ($_POST[‘sesso’] == ‘F’)

{

   echo “<font style=‘color:#FF66FF’>Ciao “. $_POST[‘nome’]. “, benvenuta nel mio sito!</font>”;

}

else

{

   echo “<font style=‘color:#0000CC’>Ciao “. $_POST[‘nome’]. “, benvenuto nel mio sito</font>!”;

}

?>
```

Ovviamente, visto che per l’esempio facciamo uso della tag “font”, dovremo ipotizzare di aver usato la **DTD Transitional** perché, come saprete, la tag “font” è deprecata, quindi se utilizzate una **DTD Strict** dovrete rendere il colore della scritta in altri modi.

Bene, abbiamo osservato il codice, e la sintassi non è difficile: abbiamo una **parola chiave**, **if**, con, tra **parentesi tonde**, la **condizione che vogliamo verificare**. Subito dopo apriamo una **parentesi graffa**, inseriamo ciò che vogliamo far compiere al preprocessore se si verifica **quella determinata condizione**, e quindi **chiudiamo la graffa**. Subito dopo scriviamo l’altra parola chiave, **else** (che possiamo tradurre con “altrimenti”), non la facciamo seguire da **nessuna condizione** tra parentesi tonde e quindi inseriamo **l’azione** sempre **tra parentesi graffe**. Ricapitoliamo:

```
if (condizione)

{

   azioni

}

else

{
```

```
   azioni

}
```

Non specifichiamo nessuna condizione dopo else per il semplice fatto che, tra le parentesi graffe che lo seguono, è specificata un'azione **per tutto ciò che non rispetta nessuna delle condizioni** che abbiamo stabilito in precedenza (in questo caso è una sola). Questo perché potremmo aver stabilito, in precedenza, **più di una condizione**. Facciamo un altro esempio e ipotizziamo di voler creare un quiz **con tre risposte** (sempre con il form che funziona tramite i *radio button*). In questo caso il codice nella pagina di arrivo del form sarà il seguente:

```
<?php



if ($_POST['risposta'] == 'A')

{

   echo "No, completamente sbagliata!";

}

else if ($_POST['risposta'] == 'B')

{

   echo "Ci sei andato vicino ma è sbagliata!";

}

else

{

   echo "Bene, risposta esatta!";

}



?>
```

Come potete notare, la seconda condizione viene verificata attraverso l'istruzione **"else if"**, che deve essere utilizzata quando tutte le condizioni sono **mutuamente esclusive** (e cioè se una è vera, tutte le altre devono essere false). Se abbiamo **sei condizioni possibili**

mutuamente esclusive, la prima sarà verificata con l'istruzione **"if"**, le quattro successive con **"else if"** e l'ultima con **"else"**.

Se al posto di "else if" inseriremo semplicemente l'istruzione "if" anche per le condizioni successive alla prima, avremo dei risultati inaspettati (vale a dire che il programma non farà quello che noi vorremo) per il fatto che **due "if" consecutivi non sono legati tra di loro** come lo sono invece un "if" e uno o più "else if". In altre parole, il blocco "corretto" si ferma quando una **condizione viene verificata**, invece il blocco "non corretto" continua a verificare tutte le operazioni. Facciamo finta di avere scritto una cosa bruttissima come questa:

```
<?php

if ($_POST['risposta'] == 'A')

{

   echo "No, completamente sbagliata! ";

}

if ($_POST['risposta'] == 'B')

{

   echo "Ci sei andato vicino ma è sbagliata! ";

}

else

{

   echo "Bene, risposta esatta! ";

}

?>
```

Che cosa succede se il nostro utente ha risposto A? Vedrà sul proprio schermo questa scritta: "No, completamente sbagliata! Bene, risposta esatta!". Perché succede questo? Il blocco, scritto in questo modo, **ha controllato la prima condizione**, che è soddisfatta e quindi viene stampata la scritta corrispondente. Ma ha controllato **anche, separatamente, la seconda condizione**, quella della risposta B! Che non può essere soddisfatta perché abbiamo utilizzato dei *radio button* e l'utente può dare una sola risposta.

Quindi cosa succede? Il programma pensa “la condizione introdotta dal secondo if non è verificata, quindi stampo **tutto ciò che è racchiuso nell’else**”. Ed ecco perché vengono stampate la prima e la terza scritta. Possiamo metterci una pezza mettendo un if($_POST[‘risposta’] == ‘C’) prima della terza scritta, ma faremo solo sprecare **“energie”** utili al nostro preprocessore che dovrà controllarsi tutte le condizioni (invece, con un blocco corretto di if-else, si fermerà non appena avrà trovato la condizione soddisfatta).

Possiamo anche avere tante condizioni e due sole azioni possibili, per esempio un quiz che ha domande con cinque risposte, e vogliamo stampare solo due scritte “Risposta esatta” o “Risposta sbagliata”. In questo caso, supponendo che la risposta esatta sia la “A”, scriveremo:

```
<?php

if ($_POST[‘risposta’] == ‘A’)

{

   echo “Risposta esatta!”;

}

else

{

   echo “Risposta sbagliata!”;

}

?>
```

Se l’utente ha risposto “A”, apparirà la scritta “Risposta esatta!”, se invece avrà risposto in qualsiasi altro modo (“B”, “C” ecc.) apparirà la scritta “Risposta sbagliata!”.

Una cosa di cui non abbiamo ancora parlato: **facciamo molta attenzione** a utilizzare **due uguali nell’espressione della condizione** e non un uguale solo, perché, come abbiamo detto nell’**articolo sui tipi**, **un uguale** è un **operatore di assegnazione**, **due uguali** sono un **operatore di confronto** (che è quello che ci serve per scrivere la nostra condizione). Se metteremo un solo uguale avremo dei risultati sgradevoli, perché il nostro blocco sì, funzionerà, nel senso che non restituirà errore, ma restituirà dei risultati molto bizzarri e **cambierà il valore di eventuali variabili** (perché un solo uguale è, ripeto, un operatore di assegnazione: è quello che si usa con le variabili come abbiamo visto qualche articolo fa).

E a proposito di operatori, oltre a quello di confronto possiamo usarne **molti altri**, ve li elenco: **maggiore** (>), **maggiore o uguale** (>=), **minore** (<), **minore o uguale** (<=), **diverso** (!=), **negativo** (!). Facciamo due esempi con il form di benvenuto che abbiamo usato all'inizio e impostiamo due condizioni, una per fare in modo che chiunque non si chiami Mario venga salutato in un certo modo e una per dare una risposta all'utente che non ha inserito il suo nome:

```
<?php



//primo esempio

if ($_POST['nome'] != "Mario")

{

   echo "Tu non ti chiami Mario!";

}

else

{

   echo "Ciao Mario!";

}



?>
```

```
<?php



// secondo esempio

if (!$_POST['nome'])

{

   echo "Perché non hai inserito il tuo nome???";
```

```
}

else

{

   echo “Ciao “. $_POST[‘nome’]. “. ti do il benvenuto sul mio sito!”;

}



?>
```

Come potete notare, l’**operatore di diversità** si usa come l’**operatore di confronto** (e come tutti gli altri **a eccezione di quello di negatività**), cioè va posto tra la variabile che vogliamo controllare e il valore che deve soddisfare la condizione. Invece l’operatore di negatività deve essere posto **subito prima della variabile** che vogliamo verificare.

Si possono anche creare dei blocchi che verifichino, **con un solo if, più di una condizione**. Lo possiamo fare utilizzando **gli operatori logici** che abbiamo visto nell’articolo sui tipi. Quindi, sempre lavorando sull’esempio precedente, possiamo controllare che il nostro utente non si chiami **né Mario né Federico**:

```
<?php



if ($_POST[‘nome’] != “Mario” && $_POST[‘nome’] != “Federico”)

{

   echo “Tu non ti chiami né Mario né Federico!”;

}

else if ($_POST[‘nome’] == “Mario”)

{

   echo “Ciao Mario!”;

}

else

{
```

```
  echo “Ciao Federico!”;

}



?>
```

Come si può notare, basta inserire l’operatore logico “&&” per **verificare insieme due condizioni**. La stessa cosa vale anche per gli altri operatori (prendiamo per esempio l’**operatore or**):

```
<?php

if ($_POST[‘nome’] == “Mario” || $_POST[‘nome’] == “Federico”)

{

  echo “Tu potresti essere o Mario o Federico!”;

}

else

{

  echo “Tu non sei né Mario né Federico!”;

}

?>
```

Infine, è possibile anche **annidare i blocchi** per creare **strutture più complesse**. Per esempio, riprendiamo il form per restituire la scritta di benvenuto colorata a seconda del sesso e aggiungiamo una domanda: chiediamo al nostro utente se è già stato sul nostro sito o no, usiamo ancora dei *radio button* e chiamiamoli “ritorno”. Modifichiamo quindi il codice per far stampare la scritta “Benvenuto” se è la prima volta che l’utente entra nel nostro sito, oppure “Bentornato” se non è la sua prima visita:

```
<?php


if ($_POST[‘sesso’] == ‘F’)
{
```

```
        echo “<font style=‘color:#FF66FF’>Ciao “. $_POST[‘nome’]. “, benvenuta nel mio sito!
</font>”;
} // questa graffa è importante: chiudiamo il primo if
else
{ //inizia l’else del primo if
} // chiudiamo tutto il blocco

?>
```

Come potete notare non cambia niente: bisogna solo **fare attenzione** a come vengono sistemate **le parentesi graffe**, perché se ve ne scordate **anche soltanto una** il programma vi restituirà **un errore**, e se invece la inserirete **nel posto sbagliato** i risultati non saranno quelli sperati. È buona norma **“indentare”** gli if annidati, per rendere **più ordinato il codice** e anche per capire bene dove iniziano e finiscono i blocchi. Quindi:

```
if (condizione)
{
}
else {
}
```

Ripeto fate attenzione perché **dimenticare di chiudere una parentesi graffa è facilissimo** e capita spesso. Non è inusuale perdere molto tempo per cercare di capire cosa non va in un programma per poi accorgersi alla fine che era tutta colpa di una parentesi graffa dimenticata! Infine, ultima cosa: in tutti gli esempi di cui sopra PHP è **case sensitive** (mi riferisco al controllo delle condizioni), per cui le parole “Federico” e “federico” sono due cose diverse.

Terminiamo il capitolo con un esercizio. Paperone vuole inserire nel sito della sua catena di alberghi un form che restituisca al cliente l’albergo di Paperopoli più adatto alle sue esigenze. Il form chiede all’utente se preferisce un hotel di lusso o uno economico e se lo vuole con piscina o senza. Gli hotel di lusso sono il Duck Palace Hotel (senza piscina) e il Grand Hotel Villa Papera (con piscina), mentre quelli economici sono l’Hotel Papero (senza piscina) e il Relais dei Paperi (con piscina). A seconda delle preferenze dell’utente, il form gli proporrà l’albergo adeguato.

10. L’istruzione switch

Questo capitolo costituisce il completamento di quello sui **blocchi condizionali if-else**: può infatti capitare, a volte, di aver a che fare con **blocchi particolarmente lunghi**, e diventa abbastanza noioso **ripetere per molte volte** le istruzioni. Prendiamo per esempio un form che, **in base a un numero inserito dall’utente**, restituisce **il corrispettivo giorno della settimana** (uno – lunedì; due – martedì; tre – mercoledì e così via). Sì, è un esempio molto stupido ma è l’ideale per capire l’argomento di questo capitolo!

Prima di tutto costruiamo il form:

```
<form action=“giorno.php” method=“post”>

Inserisci il numero per conoscere il giorno:

<input type=“text” name=“giorno” /><br /><br />

<input type=“submit” name=“invia” value=“Vai!” />

</form>
```

E adesso costruiamo il codice della pagina “giorno.php” con un blocco if, come abbiamo imparato nel capitolo precedente:

```
<?php

if ($_POST[‘giorno’] == ‘1’)
{
echo “Lunedì”;
}
else if ($_POST[‘giorno’] == ‘2’)
{
echo “Martedì”;
}
else if ($_POST[‘giorno’] == ‘3’)
{
echo “Mercoledì”;
}
else if ($_POST[‘giorno’] == ‘4’)
{
echo “Giovedì”;
}
else if ($_POST[‘giorno’] == ‘5’)
{
echo “Venerdì”;
```

```
}
else if ($_POST[‘giorno’] == ‘6’)
{
}
else if ($_POST[‘giorno’] == ‘7’)
{
}
else
{
}

?>
```

Notate che l’ultima istruzione prevede tutti **i casi diversi** da quelli che abbiamo **contemplato in precedenza**, quindi se il nostro utente inserisce nel form uno zero o un otto, per esempio, verrà restituita la scritta “Inserisci un numero compreso tra 1 e 7!”. Ma potete anche notare che questo codice **è lungo e ripetitivo**. C’è però **un modo per abbreviarlo**: utilizzando **l’istruzione “switch”**. Quest’ultima si comporta **come un blocco if** con il vantaggio di **ridurre notevolmente** il numero delle righe di codice. Mostriamo quindi come si scrive il codice che abbiamo utilizzato nell’esempio **facendo uso dell’istruzione switch**:

```
<?php

switch ($_POST[‘giorno’])
{
}

?>
```

Visto come lo abbiamo accorciato? E poi è molto semplice, come potete notare dalla sintassi. Si parte scrivendo l’istruzione **switch**, seguita dalla **variabile che vogliamo controllare** inserita **tra parentesi tonde**. Si apre poi **una parentesi graffa** all’interno della quale inseriamo **i valori** per i quali vogliamo che si **verifichi un certo evento**. I valori, racchiusi come al solito **tra apici o doppi apici**, devono essere preceduti dalla parola **“case”** e seguiti da **due punti**: dopo i due punti inseriamo **l’azione** che vogliamo che si verifichi, seguita **da punto e virgola e dall’istruzione “break;”** (tra poco vedremo qual è il suo significato).

Infine, l’ultima azione, invece della parola “case” deve essere preceduta dalla parola **“default”**: si tratta dell’azione che sarà eseguita dal programma nel caso in cui non si verifichi nessuna delle condizioni che abbiamo elencato in precedenza. Quindi avrete capito che i **“case” corrispondono a “if” e agli “else if”**, e il **“default” corrisponde a “else”**.

Dicevamo dell’istruzione **“break”**. Ecco questa serve per fare in modo che **il programma si fermi** una volta verificata una certa condizione. Cosa succede se, per esempio, non inseriamo questa istruzione dopo il primo “case” (quello che fa stampare la scritta “Lunedì” se l’utente inserisce il numero uno)? Succede questo: il programma non si ferma all’istruzione del primo “case”, ma si fermerà al **“break” immediatamente successivo**. Per cui, se avremo dimenticato il primo break e l’utente avrà inserito il numero 1, il programma di cui sopra stamperà la scritta “LunedìMartedì”, perché si è fermato al break **immediatamente successivo**.

L’azione di default può anche essere omessa (così come si può tranquillamente omettere un else in un blocco condizionale): tuttavia, così come è consigliabile prevedere sempre un else per un if, è anche consigliabile prevedere un default per uno switch. E attenzione: l’azione di default deve essere **sempre l’ultima** del nostro blocco switch, altrimenti potrebbero verificarsi azioni non gradite. Se per esempio inseriamo l’azione di default prima dell’ultimo case e poi l’utente inserisce il numero 8, non verrà stampata soltanto la scritta “Inserisci un numero ecc.”, ma anche la scritta “Domenica”, perché tutto quello che viene dopo il default **verrà interpretato**, appunto, **come default**.

Possiamo anche assegnare una azione a più case, come in questo esempio:

```
<?php

switch ($_POST[‘giorno’])
{
}

?>
```

Basterà cioè far seguire **degli spazi vuoti** ai case che vogliamo raggruppare e inserire l’azione vicina all’ultimo case del gruppo.

Ecco infine l’esercizio di questo capitolo. Utilizzando lo switch provate a costruire una calcolatrice che esegua addizioni, sottrazioni, moltiplicazioni e moduli. L’utente inserisce i numeri tramite form e il programma restituisce il risultato dell’operazione. Il form dovrà avere tre input: il primo per il primo numero, il secondo per l’operatore (più, meno, per, diviso), il terzo per il secondo numero. La soluzione nell’ultimo capitolo.

11. Cenni sulle funzioni

Nei capitoli precedenti abbiamo fatto qualche piccolo cenno alle famose **“funzioni”**, senza però mai entrare nel merito, perché ho preferito aspettare di darvi qualche strumento per poter creare non dico funzioni eccessivamente complesse, ma almeno qualcosa di **leggermente impegnativo**.

Per questo capitolo ci limiteremo a qualche esempio un po’ **“astratto”**, ma nel prossimo capitolo vedremo insieme, come ulteriore esempio, **una funzione** che potrà essere utile nei vostri siti: si tratterà di una funzione che ho già applicato con successo in alcuni miei siti e nel prossimo articolo vedremo come costruirla.

Che cos’è una funzione? È un **insieme di istruzioni**. Fin qui è semplice vero? Questa definizione già potrebbe andare bene, ma vediamo di scendere **un po’ più nel dettaglio**: una funzione è un insieme di istruzioni che **può** ricevere **uno o più parametri in ingresso** e restituisce **un risultato**. Una funzione, quando viene **invocata**, in qualsiasi punto del codice, **produrrà infatti il risultato determinato dalle istruzioni** che avremo definito all’interno della funzione stessa.

Affinché una funzione funzioni (che brutto gioco di parole!), deve essere prima **definita** (ogni funzione dovrà avere un nome), e poi invocata dove vogliamo. Facciamo quindi un esempio pratico. Negli ultimi articoli abbiamo visti i **blocchi condizionali**. Supponiamo di voler inserire nel nostro sito un form per controllare **se un anno è bisestile**. Il form avrà un campo in cui l’utente inserirà il numero dell’anno, e riceverà in risposta la scritta “Bisestile” in caso affermativo e “Non bisestile” in caso negativo.

Ricordo che un anno è bisestile se è divisibile per quattro ma non per cento (e soprattutto se è maggiore di 1582, anno a partire dal quale è stato introdotto il calendario gregoriano che utilizziamo ancora oggi).

Il form sarà molto semplice:

```
<form action=“bisestile.php” method=“post”>

<input name=“anno” type=“text” /><br />

<input type=“submit” value=“invia” /><br />
```

Andiamo ora nella pagina “bisestile.php” e definiamo la nostra funzione, che chiameremo

semplicemente “bisestile”.

```
<?php

function bisestile($anno)
{
}

?>
```

Abbiamo quindi già visto come si definisce una funzione. Si inizia scrivendo **“function”** e facendo seguire questa parola **dal nome** che vogliamo dare alla funzione (nel nostro caso “bisestile”). Subito dopo inseriamo **il parametro (o i parametri) in ingresso**, che devono essere posti **tra parentesi tonde**: nel nostro caso è una variabile che si chiama $anno. Infine, **tra parentesi graffe**, inseriamo tutte le istruzioni: nel nostro caso **un blocco if** che controlla se l’anno è bisestile o no.

Avrete intuito che la variabile **$anno** è l’anno che la funzione **riceve in ingresso** e che **dovrà essere controllato**. Per invocare la funzione basterà scrivere una formula come questa:

```
<?php

bisestile($_POST[‘anno’]);

?>
```

Abbiamo quindi **il nome della funzione** (senza la scritta “function” davanti, che ci serve **solo in fase di definizione**) e, tra parentesi, **il dato che vogliamo passare come parametro**: nel nostro caso, il dato inserito dall’utente nel form (il tutto chiuso da punto e virgola). La nostra variabile superglobale $_POST[‘anno’] sarà automaticamente trasformata nella variabile $anno e su di essa verranno eseguite le istruzioni indicate all’interno della nostra funzione. Vediamo il codice completo della pagina bisestile.php:

```
<?php
```

```
function bisestile($anno)

{

  if($anno > 1584 && $anno%4 == 0 && $anno%100 != 0)

  {

     echo “Bisestile”;

  }

  else

  {

    echo “Normale”;

  }

}



bisestile($_POST[‘anno’]);



?>
```

Le scritte verranno stampate nel punto esatto in cui inseriamo **l’invocazione della funzione**. Una funzione **può anche non avere parametri**, come in questo caso semplicissimo:

```
<?php



function hello()
{

  echo “Ciao a tutti!”;

}
```

```
?>
```

E che invocheremo così:

```
<?php

hello();

?>
```

Facciamo invece un esempio di funzione **con due parametri** e ipotizziamo un form in cui l'utente inserisca nome e cognome. La funzione seguente stamperà i dati del nostro utente:

```
<?php

function stampa($nome, $cognome)
{
   echo $nome." ".$cognome;
}

?>
```

Come vedete, i parametri devono essere separati **semplicemente con una virgola**. La stessa cosa vale **quando invochiamo la nostra funzione** in un punto qualsiasi della nostra pagina:

```
<?php

stampa($_POST['nome'], $_POST['cognome']);
```

```
?>
```

Qual è **la grande comodità** che ci forniscono le funzioni? Adesso non abbiamo avuto modo di saggiarla completamente, ma in caso di grandi “pezzi” di codice, molto lunghi, le funzioni possono servirci per **suddividere meglio i compiti** che il nostro “programma” deve eseguire, anche perché si possono inserire **funzioni all’interno di funzioni**. Modifichiamo quindi la funzione che abbiamo appena visto per far stampare anche la scritta “Benvenuto” oltre al nome e al cognome, e per farlo utilizzeremo un’altra funzione:

```
<?php


function benvenuto()

{

   echo “Benvenuto “;

}


function stampa($nome, $cognome)

{

   benvenuto();

   echo $nome, ” “, $cognome, “!”;

}


?>
```

Visto? Abbiamo invocato la funzione “benvenuto” che stampa semplicemente la scritta “Benvenuto”, e subito dopo abbiamo dato l’istruzione di stampare le due variabili $nome e $cognome (opportunamente formattate con spazi e punto esclamativo).

Un’altra comodità che ci viene data dalle funzioni è quella di **“risparmiare” codice in caso di operazioni ripetitive**. Se per esempio dobbiamo inserire un blocco if in tutte le

pagine, potremo utilizzare una funzione e poi includerla in un file esterno (che sarà opportunamente richiamato) per ottenere il risultato voluto.

C'è infine un'ultima utile istruzione da conoscere quando si parla di funzioni: l'istruzione **return**. Quest'ultima consente di interrompere l'esecuzione di una funzione e di fare in modo che il risultato della funzione possa essere salvato in una variabile. Consideriamo questo esempio:

```
<?php


function ciao()

{

   return "Ciao";

}


$ciao = ciao();


echo $ciao;


?>
```

In questo modo, otterremo la stampa della parola "Ciao", perché l'abbiamo associata a una variabile $ciao grazie alla nostra funzione: con una funzione che termina con un echo, non avremmo potuto farlo.

Consideriamo un esempio più complesso per comprendere il concetto di "interruzione dell'esecuzione": poniamo quindi di avere un sito che vende bevande alcooliche e, prima di mostrare le pagine, chiede di inserire l'età e il sesso all'utente. Nel caso in cui l'utente inserisca un'età minore di 18 anni, il sito restituirà una stringa che informerà l'utente del fatto che non può navigare oltre. In caso contrario, faremo partire un blocco if che mostra la scritta di benvenuto a seconda del sesso dell'utente:

```
<?php
```

```
function controllo_sito()
{
  if ($_POST[‘eta’] < 18)
  {
    return “Mi dispiace, ma non puoi navigare su questo sito”;
    // La funzione si interrompe qua
  }

  else
  {

    if ($_POST[‘sesso’] == ‘F’)
    {
      return “<font style=‘color:#FF66FF’>Ciao “. $_POST[‘nome’]. “, benvenuta nel mio sito!</font>”;
    }
    else
    {
      return “<font style=‘color:#0000CC’>Ciao “. $_POST[‘nome’]. “, benvenuto nel mio sito</font>!”;
    }

  }

}

?>
```

12. Un po’ di pratica

Come annunciato nell’ultimo capitolo, questo sarà **quasi puramente pratico**, nel senso che cercheremo di **applicare tutti i concetti** che abbiamo imparato fino a questo punto per costruire **uno script utile per i vostri siti** (ma impareremo anche **qualche nuovo concetto teorico** che ci sarà utile per l’esempio del capitolo e per i nostri futuri lavori con il PHP).

Quello che vi propongo qua è uno script che ho usato anche io in alcuni miei siti con successo, è impegnativo ma non è difficile da comprendere: si tratta di uno script attraverso il quale far apparire sul proprio sito delle **immagini casuali** e **abbinare a esse un testo scritto**.

Ma iniziamo subito a lavorare al nostro script. Partiamo da un caso più semplice, ovvero uno script per mostrare non due immagini casuali, **ma una sola**: lo facciamo con una sola perché è più semplice (e come esercizio per il presente capitolo… lascio a voi l’onore di creare la funzione per mostrare due immagini casuali diverse l’una dall’altra). La prima cosa da fare è creare un file che chiameremo “immagini_casuali.php” (perché non ci basterà una sola funzione e per comodità, se volessimo inserire lo script in ogni pagina del nostro sito) che includeremo, come abbiamo visto, **con il comando include**.

Apriamo ora il file “immagini_casuali.php” e **iniziamo a costruire il nostro script**. La prima cosa da fare è **creare un array delle immagini**. Abbiamo visto come creare array di stringhe, e per creare l’array con le nostre immagini sarà sufficiente **inserire il loro indirizzo** in una stringa di testo. Poniamo quindi di avere tre immagini, “prima.jpg”, “seconda.jpg” e “terza.jpg” e di averle inserite nella cartella “Grafica”. Inizializziamo un array “canonico”, con **indici numerici** e partendo da zero (potete dichiarare l’array nel modo che preferite):

```
$immagini[] = “Grafica/prima.jpg”;

$immagini[] = “Grafica/seconda.jpg”;

$immagini[] = “Grafica/terza.jpg”;
```

Ecco che abbiamo creato un array chiamato “$immagini” e lo abbiamo riempito con le nostre tre fotografie o disegni. La prossima cosa da fare è **trovare l’indice dell’ultimo elemento dell’array**. Per farlo utilizziamo un “barbatrucco”, ovvero **contiamo gli elementi** del nostro array e **diminuiamo il valore ottenuto di una unità**. Perché questo? Lo abbiamo visto nel capitolo sugli array: **gli elementi sono tre**, ma siccome gli indici **partono da zero**, l’indice dell’**ultimo elemento** sarà **due**. Per contare gli elementi dell’array esiste **una funzione predefinita** in PHP, che si chiama **“count”**: riceve come

parametro **il nostro array** e restituisce proprio **il numero degli elementi**. Molti avranno quindi già intuito come ricavare l'indice dell'ultimo elemento:

```
$ultimo_elemento = count($immagini) - 1;
```

Abbiamo quindi creato una variabile chiamata **"$ultimo_elemento"** attraverso la funzione *count*, che ci restituisce il numero degli elementi dell'array, e abbiamo diminuito il tutto di una unità. Quindi adesso il valore associato a "$ultimo_elemento" è 2. Adesso dobbiamo creare una variabile alla quale **assegnare un indice casuale**, e ci servirà quando **faremo apparire** l'immagine casuale. Ovviamente assegnare un indice casuale significa sostanzialmente **trovare un numero a caso**, e anche per questo compito **esiste una funzione predefinita** in PHP: si tratta della funzione **"rand"**, che riceve **due parametri**, ovvero due numeri (che rappresentano **l'intervallo** all'interno del quale cercare un numero a caso) e restituisce come risultato proprio **un numero casuale**. Dal momento che abbiamo tre immagini e l'indice dell'elemento più grande del nostro array è 2, noi vogliamo scegliere un numero casuale tra 0 e 2:

```
$casuale = rand(0, $ultimo_elemento);
```

Con questa riga di codice abbiamo creato una variabile "$casuale" per mezzo della funzione "rand", che restituisce un numero a caso. Pertanto non possiamo sapere quale valore avrà la funzione (se 0, 1 o 2): ce ne accorgeremo solo quando l'immagine apparirà nel sito.

Ecco che abbiamo quindi terminato le operazioni preliminari. Possiamo già fare in modo che sul nostro sito **appaia l'immagine**, e lo facciamo **creando una funzione** che chiameremo **"rotazione"**. All'interno di questa funzione inseriremo semplicemente **una stampa dell'immagine casuale**, ma prima dovremo rendere **globali** le variabili $immagini, $ultimo_elemento e $casuale. Che cosa sono **le variabili globali**? Questo è un **concetto importante** che impariamo in questo capitolo: si tratta di variabili **dichiarate esternamente** che possono essere utilizzate **anche nelle funzioni**. Infatti una variabile dichiarata fuori da una funzione, come nel nostro caso, non è utilizzabile all'interno di funzione: sarà necessario **renderla globale**. Per farlo sarà sufficiente, all'interno della funzione, inserire la parola **"global"** davanti al nome della variabile. Quindi, dopo aver dichiarato le variabili globali, stampiamo la nostra immagine:

```
function rotazione()

{
```

```
  global $immagini;

  global $ultimo_elemento;

  global $casuale;

  echo “<img src=’{$immagini[$casuale]}’ alt=‘Immagine’ />”;

}
```

Ho inserito un “alt” generico per comodità. Manca ancora un ultimo passaggio: per fare in modo che l’immagine appaia, dobbiamo richiamare la funzione **nel punto del codice** in cui vogliamo **mostrare l’immagine**, e lo facciamo molto semplicemente in questo modo:

```
<?php

rotazione();

?>
```

Prima di continuare è doveroso fare una **precisazione**. Nessuno ci vieta di inserire, nella stringa, tutto il codice relativo all’immagine (e magari anche un piccolo testo), per esempio così:

```
$immagini[] = “<img src=‘Grafica/prima.jpg’ alt=‘Prima immagine’ /><br />Prima immagine”;
```

Ma cosa succede se, tra l’immagine e il testo, dovessimo frapporre una qualsiasi altra cosa che **non possiamo determinare nella funzione**? Il primo esempio che mi salta in testa: una immagine, un testo non correlato all’immagine e alla fine una nota che fornisce spiegazioni sull’immagine. Bisogna modificare la funzione per fare in modo che **all’immagine venga abbinato un testo**. In realtà si tratta di una operazione **piuttosto semplice**, perché basterà creare una nuova funzione che chiameremo **“associazione”** e attraverso la quale faremo in modo che, dato un indice dell’array di immagini, a esso venga associato un testo.

Come eseguiamo questa operazione? Con una serie di **if** o, se preferite, con uno **switch**: il blocco dovrà controllare la **variabile globale $casuale** e quindi associare a essa il testo rispettivo. Costruiamo quindi la nostra funzione:

```
function associazione()
{
}
```

A questo punto stampiamo l’immagine semplicemente scrivendo **rotazione()**; in un punto qualsiasi del codice, e in un altro punto facciamo la stampa del testo scrivendo **associazione()**; e se avremo fatto tutto bene vedremo **da una parte la nostra immagine e dall’altra il testo abbinato**!

Ricapitoliamo quindi tutto il codice del nostro script:

```
<?php

// creiamo l’array delle immagini
$immagini[] = “Grafica/prima.jpg”;
$immagini[] = “Grafica/seconda.jpg”;
$immagini[] = “Grafica/terza.jpg”;

// troviamo l’ultimo elemento dell’array
$ultimo_elemento = count($immagini) – 1;

// estraiamo a sorte un numero
$casuale = rand(0, $ultimo_elemento);

// funzione per stampare l’immagine casuale
function rotazione()
{
  // rendiamo globali le variabili
  global $immagini;
  global $ultimo_elemento;
  global $casuale;
  // stampiamo l’immagine
  echo “<img src=’{$immagini[$casuale]}’ alt=‘Immagine’ />”;
}

// funzione per abbinare testo all’immagine casuale
function associazione() {
```

```
  // rendiamo globale la variabile

  global $casuale;

  // con uno switch abbiniamo il testo al numero dell'indice
}


?>
```

Ed ecco quindi che abbiamo creato un simpatico e semplice script per **stampare immagini casuali e abbinare a esse un testo**, utilizzando più o meno tutto ciò che abbiamo imparato finora e imparando anche alcune cosine nuove.

Come vi anticipavo in apertura, l'esercizio di questo capitolo consiste nel modificare lo script in modo da stampare due immagini e fare sì che non siano due immagini uguali ma siano l'una diversa dall'altra!

13. I controlli iterativi: while e for

Ricordate quando, nel capitolo sugli array, avevamo detto che esiste **un metodo molto più rapido** per stampare **gli elementi di un array**? Bene, come ricorderete, avevamo stampato i singoli elementi ognuno con il proprio comando *echo*. Ma avevamo detto che nel caso di array con **decine di elementi** e per i quali non sappiamo quali elementi corrispondano a quali indici, quel metodo è **totalmente inservibile**.

Ho utilizzato questo esempio per introdurre i **controlli iterativi** perché possiamo così vederli "agire" in modo pratico. Ma cosa sono i controlli iterativi? La parola **"iterativo"** deriva dal **latino**, e più precisamente dall'avverbio *iterum*, che significa **"di nuovo"**: infatti questi controlli non fanno altro che ripetere **la stessa operazione** per un numero di volte che avremo specificato prima di far partire il controllo. Questi controlli possono essere chiamati anche **"cicli"** perché la loro natura è, appunto, quella di compiere **ciclicamente la stessa operazione**.

Ma veniamo all'esempio del nostro array e utilizziamo proprio quello che avevamo usato nel capitolo sugli array:

```
<?php



$array[] = "Federico";

$array[] = "Ilaria";

$array[] = "Gabriele";



?>
```

Questo è il nostro array, e adesso vedremo da vicino come utilizzare i cicli per **stamparne gli elementi**. In PHP esistono quattro controlli iterativi: **while, do-while, for e foreach**. In questo capitolo vedremo i primi tre, mentre a foreach dedicheremo il prossimo.

Iniziamo da **while**. La parola in inglese significa, tra le altre cose, anche **"finché"**, e questo ci aiuta a comprendere la natura del controllo: ovvero **compiere una operazione finché è verificata una certa condizione** (e la condizione potrebbe essere nient'altro che il numero delle volte che vogliamo che tale operazione venga eseguita).

La struttura dei cicli while è **simile** a quella dei blocchi condizionali. Abbiamo la parola chiave (*while*) seguita dalla **condizione tra parentesi tonde** e, **tra parentesi graffe**, **le operazioni da eseguire**. Nel caso del nostro array dovremo però dichiarare prima un paio di variabili, perché dovremo far eseguire l'operazione di stampa dell'elemento tre volte (visto che nel nostro array abbiamo tre elementi). Come si fa? Abbiamo bisogno di un **contatore**, che dovrà contare il numero delle operazioni compiute dal ciclo, e di una variabile che ci dica **quando fermarci**.

Il contatore solitamente si chiama **$i** (ma potete chiamarlo come volete, però è un "classico" chiamarlo **$i**) e va impostato a 0, mentre la variabile che ci dirà quando fermarci non sarà altro che **la lunghezza del nostro array**, che otterremo, come già sappiamo, tramite la funzione **count**:

```
$i = 0;

$lunghezza = count($array);
```

Dichiarate queste due variabili possiamo **dare inizio al nostro ciclo**, e il meccanismo è questo: ogni volta che un'operazione viene compiuta, **il contatore viene incrementato** (abbiamo visto come si fa nell'articolo sulle operazioni con le variabili, ricordate?) e si va avanti finché il valore del contatore rimane **minore rispetto alla lunghezza dell'array**. Per far stampare l'elemento non facciamo altro che **abbinare** all'indice del nostro array **il valore** che il contatore assume **in quel momento**. Ma forse vedendo il codice potremo capire meglio il tutto:

```
while ($i<$lunghezza)

{

   echo $array[$i]. "<br />";

   $i++;

}
```

Allora, dicevamo che dopo la parola chiave (while) abbiamo **la condizione**: in questo caso, la condizione è "finché $i, cioè il contatore, rimane minore della lunghezza dell'array". Tra **parentesi graffe** abbiamo **le istruzioni**, quindi "stampa l'elemento dell'array che ha l'indice uguale a $i, cioè al valore del contatore, andando a capo con un break html concatenato, e poi incrementa il contatore".

Al primo giro, con $i uguale a 0, viene stampato l'elemento $array[0], cioè "Federico", e la variabile $i viene incrementata e diventa 1, poi al secondo giro abbiamo $i uguale a 1 e verrà stampato $array[1], cioè Ilaria, e così via fino all'ultimo. Quando arriviamo a Gabriele il valore di $i passa da 2 a 3: dal momento che adesso $i non è più minore di $lunghezza, perché il valore di entrambe ora è 3, la condizione $i<$lunghezza non è più soddisfatta e il nostro controllo è terminato.

È possibile ottenere **lo stesso risultato** anche con **gli altri due controlli iterativi**. Il controllo **do-while** è del tutto simile a quello che abbiamo visto prima: la differenza consiste nel fatto che **condizione e istruzioni sono "invertite"**. Si introduce infatti una nuova parola chiave, **"do"**, subito dopo la quale andranno **aperte le graffe** e, dopo la graffa chiusa, andrà subita inserita la parola chiave che già conosciamo, **"while"**, con **la condizione tra parentesi tonde**:

```
do
{

   echo $array[$i], "<br />";

   $i++;

}

while ($i<$lunghezza);
```

Ricordiamoci di inserire il **punto e virgola** dopo la parentesi tonda chiusa. E facciamo anche molta attenzione a **non inserire niente** (commenti a parte) tra il "do" e la graffa aperta e tra la graffa chiusa e il "while": il risultato, qualora inserissimo una qualsiasi cosa che non sia un commento in questi punti, sarà un inevitabile errore.

Ma veniamo ora al ciclo **for**, che è un pochino diverso rispetto a while e a do-while: il **grande vantaggio** che for ha rispetto ai primi due consiste nel fatto che **non dobbiamo** dichiarare la variabile del contatore **fuori dal ciclo**, ma la dichiariamo all'interno del nostro controllo iterativo.

Lo schema è **simile a quello di while**, ovvero parola chiave seguita da **condizione tra parentesi tonde** e **istruzioni tra parentesi graffe**: cambia però il fatto che la condizione ha bisogno di **tre argomenti**: **il primo** viene valutato **una volta sola all'inizio del controllo**, **il secondo** viene valutato **finché non è soddisfatto** e **il terzo** è **una istruzione** che viene eseguita **alla fine di ogni ciclo**. Ne consegue che solitamente, nella stragrande maggioranza dei casi, il primo argomento è l'inizializzazione del contatore, il secondo è la vera condizione e il terzo è l'incremento.

Ma vedendo il controllo in azione potremo capire meglio. Dobbiamo sapere che per il for, come anticipato, non serve dichiarare la variabile $i, perché questa dichiarazione sarà nient'altro che il primo argomento del nostro for. Eccolo qua:

```
for ($i=0; $i<$lunghezza; $i++)

{

    echo $array[$i], "<br />";

}
```

Apriamo quindi il nostro controllo iterativo con la parola chiave **for**, e **tra parentesi tonde** inseriamo le tre istruzioni: la prima, $i = 0, è la dichiarazione del nostro contatore; la seconda, $i<$lunghezza, è la stessa che abbiamo visto prima con while e do-while; e infine la terza, $i++, è, come sappiamo, **l'incremento**, anch'esso **da includere nella condizione** e quindi da **non aggiungere nelle istruzioni**.

Dei tre controlli iterativi, quello che preferisco è **for**: è il più rapido, il più immediato e credo anche il più facile da comprendere malgrado sia più articolato rispetto agli altri. Dobbiamo però sapere che si tratta di tre modi diversi per dire la stessa cosa. E allora, vi chiederete, perché in PHP esistono tre modi diversi per dire una cosa sola? L'informatica non dovrebbe essere una cosa semplice e univoca?

In realtà ci sono delle **lievi differenze** che **dovrebbero condizionare** la scelta di uno dei nostri tre controlli, una scelta che dovrebbe cadere **in base alla natura di ciò che dobbiamo fare**.

Cominciamo dal capire la differenza tra while e do-while. Il fatto che nel do-while la condizione venga **controllata alla fine del ciclo** pone una discriminante, nel senso che la prima operazione, qualunque essa sia, **verrà comunque eseguita**, anche se la condizione non è soddisfatta: questo non avviene con while, dal momento che la condizione **è posta prima delle istruzioni**, e se non è soddisfatta fin dall'inizio il controllo **non si mette neanche in moto**. Ma facciamo un esempio pratico. Scrivere questo:

```
$i = 1;
while ($i<1)
{
}
```

E scrivere questo:

```
$i = 1;
do
{
}
while ($i<1);
```

Potrebbe sembrare in apparenza la stessa cosa. Ma in realtà non è così, perché nel primo caso il ciclo non stampa niente. Nel secondo invece stampa “1”, proprio perché viene comunque fatto un “giro” prima del termine. Quindi bisognerebbe scegliere do-while rispetto a while se si vuole far compiere al controllo **almeno un giro**. Il for invece ha il vantaggio rispetto a while di rendere il **codice più compatto**: quindi se si hanno tanti cicli è consigliabile usare for per non avere troppi contatori sparpagliati per il codice.

Quando si fa uso dei controlli iterativi è necessario prestare attenzione ai **“loop infiniti”**, che si creano quando facciamo in modo che una condizione **sia soddisfatta all’infinito**. Per esempio, dichiarando una variabile $j = 2:

```
for ($i=0; $i<$j; $i++)
{
  $j++;
}
```

Questo è un loop infinito. Fortunatamente PHP ha **un limite di tempo** scaduto il quale l’esecuzione di uno script **verrà chiusa in modo forzato**, quindi la macchina non vi si impallerà, però state comunque attenti perché a volte capita, magari per disattenzione, di creare dei loop infiniti.

Infine, chiudiamo con due istruzioni utili per **fermare o un ciclo o tutto il controllo**. La prima è l’istruzione **“continue”**, che ci permette di **saltare un ciclo e passare immediatamente a quello successivo**. Tipicamente si utilizza quando si trova un certo valore all’interno del controllo. Nell’esempio seguente facciamo in modo che quando la nostra struttura trova i nomi “Ilaria” e “Federico”, al posto del nome venga lasciato uno spazio bianco. Lo si fa per mezzo dell’istruzione “continue”:

```
for ($i=0; $i<$lunghezza; $i++)
{
}
```

Basta quindi semplicemente eseguire un controllo condizionale su un elemento, far eseguire istruzioni (nel nostro caso la stampa del break html) e quindi, tramite l'istruzione **"continue"**, dire al controllo di fermare l'attuale ciclo e di passare al successivo: in questo modo tutto ciò che segue il "continue" **non sarà fatto** (nel nostro caso, la stampa dell'elemento dell'array).

Nel prossimo esempio invece **fermiamo il ciclo** dopo che abbiamo trovato l'elemento "Federico", e lo facciamo con l'istruzione **"break"**:

```
for ($i=0; $i<$lunghezza; $i++)
{
}
```

La sintassi è **del tutto identica** a quella dell'istruzione **"continue"**: appena il controllo trova l'elemento che soddisfa la condizione che ha, tra le istruzioni, il nostro "break", immediatamente si ferma tutto.

Concludiamo il capitolo con un esercizio simpatico che ho chiamato "torta di compleanno"! Un utente ha un form in cui inserire un numero. Dato il numero, costruite un ciclo for che cucina una torta di compleanno con strati di panna e cioccolato uguali al numero inserito dall'utente, con due condizioni: sulla cima devono esserci le candeline e il primo e l'ultimo strato devono essere di panna. Per esempio, se l'utente scrive 3, il risultato sarà questo:

```
| | | |
- - -
***
- - -
***
- - -
***
- - -
```

Divertitevi!

14. I cicli foreach

Abbiamo accennato a questo argomento nell'ultimo capitolo, e abbiamo detto che ci avremmo dedicato un capitolo a parte del nostro corso: bene, è arrivato quindi il momento di affrontare i **controlli foreach**!

Foreach, così come **while, do-while e for**, è un **controllo iterativo**, quindi come sappiamo non fa altro che eseguire un certo blocco di **operazioni** finché viene **soddisfatta una condizione**, ma con una **grossa differenza** rispetto ai cicli che abbiamo visto nel capitolo precedente: *foreach* è infatti stato pensato **appositamente per gli array**. Questo significa che il nostro controllo *foreach* **scorre** un array (oppure **"percorre"** un array) e per **ognuno dei suoi elementi** compie **alcune operazioni**.

Ma cominciamo a fare qualche esempio pratico, e torniamo quindi all'array che avevamo visto nel capitolo dedicato proprio agli array:

```
<?php



$array[] = "Federico";

$array[] = "Ilaria";

$array[] = "Gabriele";



?>
```

Il controllo *foreach* può aiutarci, per esempio, a stampare **ognuno di questi elementi**, magari in colonna. Vediamo quindi l'esempio fatto su questo array per conoscere anche la sintassi del controllo *foreach*:

```
<?php



foreach ($array as $componente)

{

   echo "$componente<br />";
```

```
}


?>
```

La sintassi è quindi piuttosto semplice e intuitiva. Scriviamo la parola chiave **foreach** seguita da **un'espressione tra parentesi tonde**: questa espressione sarà del tipo **$array as $elemento**. La prima variabile è ovviamente **il nostro array**, mentre la seconda **sarà creata dal controllo** e a essa saranno associati **i valori degli elementi**, ognuno nel momento in cui viene esaminato dal controllo.

Ciò significa che al primo giro, $componente corrisponderà a "Federico", al secondo giro corrisponderà ad "Ilaria" e così via, e con la nostra istruzione non facciamo altro che far stampare al programma **ognuno dei nostri elementi**. Abbiamo quindi visto, tra l'altro, **il modo più veloce in assoluto** per stampare gli elementi di un array!

Quindi, in sostanza, si intuisce che le operazioni che noi specifichiamo **tra le parentesi graffe**, saranno eseguite **tante volte quanti sono gli elementi dell'array**, e che la variabile $componente assume un valore diverso a ogni giro.

Possiamo utilizzare il controllo foreach anche per **cambiare i valori dell'array**, con un paio di accorgimenti. Per esempio, vogliamo fare in modo che di fianco ai nostri autori venga stampata la scritta "Sito web" e vogliamo farlo **modificando i valori dell'array**.

Per prima cosa, diamo inizio a un nuovo controllo iterativo *foreach* simile a quello che abbiamo visto in precedenza, ma con una piccolissima differenza:

```
foreach ($array as &$componente)
{

  $componente = "$componente – Sito web";

}
```

Visto? Dopo la parola chiave **"as"** abbiamo inserito una **e commerciale** (ampersand) davanti alla nuova variabile $componente. Quando infatti eseguiamo un controllo foreach, la variabile chiamata $componente non è altro che **una copia del nostro array di partenza**, che rimane inalterato qualsiasi cosa decidiamo di fare sulla variabile

$componente. Con la e commerciale anteposta, facciamo invece in modo che la variabile $componente **vada a sostituire** l'elemento dell'array $array: insomma, con questo accorgimento **non si tratta più di una copia**, ma dell'elemento stesso dell'array.

Se proviamo quindi a fare una **stampa** del primo elemento, usando per esempio il comando echo $array[0], vedremo che il programma stamperà la scritta "Federico – Sito web". Per stampare gli elementi tutti assieme, non dovremo far altro che chiamare **un nuovo controllo foreach**:

```
foreach ($array as $nuovo)
{

   echo "$nuovo<br />";

}
```

Avremo quindi gli elementi dell'array con i loro nuovi valori stampati tutti in colonna.

Possiamo poi utilizzare il nostro *foreach* anche per gli array associativi. Supponiamo quindi di avere un array per un'opera d'arte come questo:

$opera = array('Opera' => 'David', 'Autore' => 'Michelangelo', 'Datazione' => '1501-1504');

Non è altro che un **array con tre elementi**, il primo che ha come indice "Opera", il secondo "Autore" e il terzo "Datazione", e li abbiamo dichiarati con il metodo della **"freccia" costruita con uguale e parentesi uncinata chiusa** (lo avevamo accennato nel capitolo sugli array per inizializzare un array e farlo partire da un indice ben preciso).

Per stampare gli elementi dovremo utilizzare il nostro *foreach* in questo modo:

```
foreach ($opera as $indice => $valore)
{

   echo "$indice: $valore <br />";

}
```

Quindi la modalità è del tutto simile a quelle che abbiamo visto finora, con la differenza che **la coppia indice-valore** dovrà essere scritta **con la "freccia"** (dopo la **parola chiave as**) nello stesso modo in cui abbiamo dichiarato l'array.

Possiamo anche **cambiare i valori di un array associativo** utilizzando lo stesso accorgimento che abbiamo visto in precedenza… per esempio, se volessimo inserire un break html in fondo alla stringa:

```
foreach ($opera as $indice => &$valore)
{

  $valore = "$valore<br />";

}
```

E poi stampiamo come abbiamo visto prima:

```
foreach ($opera as $indice => $valorenuovo)
{

  echo "$indice: $valorenuovo<br />";

}
```

Attenzione! Si possono cambiare **soltanto i valori e non gli indici**. Se provaste a mettere la e commerciale davanti all'indice, ci sarebbe un errore.

Nel capitolo sugli array, non abbiamo detto che possono esistere anche **"array di array"**, ovvero **array i cui elementi sono altri array**. Vediamo adesso un "array di array" (nel nostro caso di opere d'arte), e vediamo anche come stamparlo. Questo è l'array:

```
$opere = array

(array('Opera' => 'David',

'Autore' => 'Michelangelo',

'Datazione' => '1501-04'),

array('Opera' => 'Camera degli Sposi',
```

```
‘Autore’ => ‘Andrea Mantegna’,

‘Datazione’ => ‘1465-74’),

array(‘Opera’ => ‘Il Tramonto’,

‘Autore’ => ‘Giorgione’,

‘Datazione’ => ‘1506-08’),

array(‘Opera’ => ‘San Giorgio’,

‘Autore’ => ‘Donatello’,

‘Datazione’ => ‘1416’)

);
```

Vi consiglio di scriverlo così incolonnato, almeno è più semplice da vedere e da intuire. La sintassi comunque è questa:

```
$nome_array = (array(‘indice’ => ‘valore’), array(‘indice’ => ‘valore’), array(‘indice’ => ‘valore’));
```

Bene, per stampare il nostro array di array creeremo questo controllo:

```
foreach ($opere as $elemento)

{

   echo “<i>Opera</i>: {$elemento[‘Opera’]}<br />”;

   echo “<i>Autore</i>: {$elemento[‘Autore’]}<br />”;

   echo “<i>Datazione</i>: {$elemento[‘Datazione’]}<br /><br />”;

}
```

Alla variabile $elemento viene associato come valore **ognuno degli array presenti nel nostro “array di array”**: ne consegue che stamperemo gli elementi con la stampa tipica degli array associativi (includendo quindi l’elemento tra parentesi graffe), come abbiamo visto nel capitolo sugli array.

Terminiamo il capitolo con un esercizio per applicare con un esempio pratico il controllo foreach. Supponiamo di gestire un sito di appassionati di sci e di voler inserire in un box i risultati in diretta dei mondiali di sci, limitandoci però alle prime cinque posizioni proprio per dare un rapido flash ai nostri visitatori e per essere i primi a informarli.

La gara di slalom gigante femminile è stata vinta da Tina Maze, che ha preceduto, nell'ordine, Federica Brignone, Tessa Worley, Denise Karbon e Viktoria Rebensburg. Attraverso un controllo *foreach* stampate la classifica finale della gara in questa forma:

1. Tina Maze
2. Federica Brignone
3. Tessa Worley
4. Denise Karbon
5. Viktoria Rebensburg

NON VALE fare cose di questo tipo:

```
$array[] = "1. Tina Maze<br />";
```

Ogni elemento dell'array deve contenere solo il nome dell'atleta, così:

```
$array[] = "Tina Maze";
```

15. Un esempio pratico: controlliamo un form

A questo punto del nostro ebook, prima di procedere oltre, è bene fare un **riepilogo** di quanto appreso finora, e facciamo questo riepilogo con un esempio pratico: **creiamo un form**, lo **controlliamo** e **stampiamo i dati inseriti dall'utente**. E come sempre, a conclusione del capitolo, un esercizio per fissare i concetti appresi (perché malgrado sia un esempio di riepilogo, impareremo anche qualcosa di nuovo!).

Ricordate il form che avevamo costruito nell'esercizio sul capitolo delle **variabili relative al web**? Benissimo, **prendiamo proprio quel form** come esempio e **cerchiamo di migliorarlo**. Ricreiamo quindi il nostro form di **prenotazione per l'albergo**, con un campo per il nome, uno per il cognome, uno per l'email, uno per il numero di telefono, uno per il numero delle notti, uno per la data di arrivo, uno per il tipo di camera e infine due radio button per chiedere al nostro cliente se vuole o no la colazione:

```
<form action="prenotazione.php" method="post" name="prenotazione">

Nome:<br />

<input type="text" name="nome" /><br />

Cognome:<br />

<input type="text" name="cognome" /><br />

E-mail:<br />

<input type="text" name="email" /><br />

Telefono:<br />

<input type="text" name="telefono" /><br />

Numero notti: <br />

<input type="text" name="notti" /><br />

Data di arrivo:<br />

G: <input type="text" name="gg" size="2" value="gg" />

M: <input type="text" name="mm" size="2" value="mm" />

A: <input type="text" name="aaaa" size="4" value="aaaa" /><br />

Tipo di camera: <br />

<select name="camera">
```

```
<option value="Singola">Singola</option>

<option value="Doppia">Doppia</option>

<option value="Matrimoniale">Matrimoniale</option>

<option value="Tripla">Tripla</option>

</select><br />

Colazione: <br />

Sì <input type="radio" name="colazione" value="Con colazione" /><br />

No <input type="radio" name="colazione" value="Senza colazione" /><br /><br />

<input type="submit" name="invia" value="Invia!" /><br />

</form>
```

Con l'esercizio nel capitolo sulle variabili relative al web **non ci siamo preoccupati** di rendere il nostro form **particolarmente raffinato**: ci siamo accontentati di stampare, con alcuni *echo* messi al posto giusto, **i dati inseriti dal nostro utente**. Ma se l'utente non inserisce niente nel form e fa comunque clic sul campo "invia"? La cosa **non avrebbe molto senso**, quindi inseriamo dei **controlli sui campi** del modulo per **"costringere"** l'utente a riempire il form **in modo significativo**.

Questa operazione è possibile inserendo dei **controlli condizionali** nella pagina "prenotazione.php". Non c'è bisogno di dare ulteriori spiegazioni sui controlli condizionali, quindi vediamo direttamente il codice, e nella fattispecie vediamo come controllare che la lunghezza del nome e del cognome non sia inferiore ai due caratteri, che il numero di telefono e il numero delle notti siano effettivamente dei numeri, che la data sia scritta bene (evitando quindi arrivi il 30 di febbraio o il 56 di agosto) e che vengano specificati correttamente il tipo di camera e la colazione.

Per prima cosa impariamo **una nuova funzione**, la funzione **isset**: nelle scorse lezioni abbiamo visto come in PHP esistano delle **funzioni predefinite**, e **isset** è una di queste. Il compito della funzione è molto semplice: determina **se una variabile è stata inizializzata**, e restituirà il booleano **true in caso affermativo** e, viceversa, **false in caso negativo**.

Utilizziamo la funzione **isset**, che riceve come parametro la variabile che vogliamo controllare, per verificare se l'utente **ha fatto clic sul pulsante "invia"**: in caso affermativo **faremo tutti i controlli sul form e faremo la stampa**, in caso contrario scriveremo semplicemente "Non è stato compilato il form per la prenotazione". Il codice

sarà quindi questo:

```
if (isset($_POST['invia']) == true)

{

   // istruzioni che vedremo tra poco

}

else
{

   echo 'Non è stato compilato il form per la prenotazione.';

}
```

Il controllo condizionale verifica quindi se il risultato della funzione **isset** è uguale a *true*, e in caso affermativo eseguirà le istruzioni che vedremo adesso, mentre in caso contrario stamperà la scritta che avvisa l'utente della mancata compilazione del form. È un controllo che ci serve se l'utente **arriva direttamente nella pagina "prenotazione.php"** senza aver correttamente compilato il form. Prima di andare avanti però dobbiamo sapere che scrivere

```
if (isset($_POST['invia']) == true)
```

equivale a scrivere

```
if (isset($_POST['invia']))
```

Proprio perché se la variabile esiste il risultato sarà sempre positivo e quindi non ci sarà bisogno di specificarlo (bisogna invece sempre specificare "false" quando vogliamo controllare se una variabile non esiste). Infine un'ultima notazione su **isset**: funziona **soltanto con le variabili**, quindi qualsiasi altro parametro passerete alla funzione (per esempio un'altra funzione) restituirà un errore.

Vediamo quindi come **controllare i singoli campi**. Per quanto riguarda tipo di camera, colazione ed email possiamo utilizzare di nuovo la funzione isset. Poi ci sarebbe anche il sistema per controllare se un indirizzo e-mail è scritto correttamente, ma non lo vediamo

qua perché richiede la conoscenza delle espressioni regolari (argomento che affronteremo nell'ultimo capitolo dell'ebook).

Per controllare invece che nome e cognome siano lunghi almeno due caratteri utilizzeremo la funzione **strlen**: riceve come parametro **la variabile da controllare** e ne restituisce **la lunghezza**, quindi ci basterà impostare **un if che controlli che la lunghezza della nostra stringa sia di almeno due caratteri**.

Per controllare invece se il numero di telefono è effettivamente un numero ci avvarremo della funzione **is_numeric**, che funziona un po' come **isset**: controlla se la variabile ricevuta come parametro **è un numero** e in caso affermativo **restituisce true**. Questo ovviamente vale per numeri di telefono che sono scritti **senza caratteri speciali**, per esempio la *slash* tra prefisso e numero: se vogliamo far scrivere all'utente il numero di telefono in questo formato dovremo, come nel caso delle email, utilizzare le espressioni regolari. Utilizziamo la funzione **is_numeric** anche per controllare **se la data è scritta bene**, e attraverso degli **if che controllano i giorni e i mesi**, verificheremo che nessuno arrivi il 30 di febbraio.

Bene, è quindi ora di vedere cosa inserire nel blocco che abbiamo lasciato in sospeso prima, e lo vediamo con il blocco intero, quindi utilizziamo il codice che abbiamo visto in precedenza e riempiamolo a dovere:

```
<?php



if (isset($_POST['invia']))

{



  // CONTROLLIAMO IL NOME

  if (strlen($_POST['nome']) < 2)

  {

    echo "<strong>Errore</strong>: Il nome deve essere di almeno due caratteri!<br />";

  }
```

```
// CONTROLLIAMO IL COGNOME

else if (strlen($_POST['cognome']) < 2)

{

   echo "<strong>Errore</strong>: Il cognome deve essere di almeno due caratteri!<br />";

}


// CONTROLLIAMO L'EMAIL

else if (isset($_POST['email']) == false)

{

   echo "<strong>Errore</strong>: non hai specificato l'email!<br />";

}


// CONTROLLIAMO IL TELEFONO

else if (is_numeric($_POST['telefono']) == false)

{

   echo "<strong>Errore</strong>: il campo <i>Telefono</i> deve contenere soltanto numeri!<br />";

}


// CONTROLLIAMO IL NUMERO DELLE NOTTI

else if (is_numeric($_POST['notti']) == false)

{

   echo "<strong>Errore</strong>: il campo <i>Notti</i> deve contenere soltanto numeri!<br />";

}
```

```
// CONTROLLIAMO CHE LA DATA SIA INSERITA

else if ((is_numeric($_POST['gg']) == false) || (is_numeric($_POST['mm']) == false) ||

(is_numeric($_POST['aaaa'])) == false  )

{

   echo "<strong>Errore</strong>: non hai specificato una data!<br />";

}



// CONTROLLIAMO GIORNI E MESI

else if ($_POST['gg'] > 31 || $_POST ['mm'] > 12)

{

   echo "<strong>Errore</strong>: data inesistente!<br />";

}



// CONTROLLIAMO I MESI CON 30 GIORNI

else if ($_POST['gg'] > 29 && ($_POST ['mm'] == 4 || $_POST['mm'] == 6 || $_POST['mm'] == 9 ||
$_POST['mm'] == 11))

{

   echo "<strong>Errore</strong>: data inesistente!<br />";

}



// CONTROLLIAMO GLI ANNI BISESTILI

 else if ($_POST['gg'] > 29 && $_POST ['mm'] == 2 && ($_POST['aaaa']>1584 &&
$_POST['aaaa']%4 == 0 && $_POST['aaaa']%100 !=0))

{

   echo "<strong>Errore</strong>: data inesistente!<br />";

}
```

```
// CONTROLLIAMO GLI ANNI NON BISESTILI

else if ($_POST[‘gg’] > 28 && $_POST [‘mm’] == 2 && ($_POST[‘aaaa’]%4 != 0))

{

   echo “<strong>Errore</strong>: data inesistente!<br />”;

}


// CONTROLLIAMO IL TIPO DI CAMERA

else if (isset($_POST[‘camera’]) == false)

{

   echo “<strong>Errore</strong>: non hai specificato il tipo di camera!<br />”;

}


// CONTROLLIAMO LA COLAZIONE

else if (isset($_POST[‘colazione’]) == false)

{

   echo “<strong>Errore</strong>: non hai specificato la colazione!<br />”;

}

else

{

   // STAMPIAMO IL RIEPILOGO!

   echo “Salve “. $_POST[‘nome’]. ” “. $_POST[‘cognome’]. “. questo è il riepilogo dei suoi dati:<br />
<br

/>”;
```

```
    echo “Titolare della prenotazione: “. $_POST[‘nome’]. ” “. $_POST[‘cognome’]. “<br />”;

    echo “E-mail: “. $_POST[‘email’]. “<br />”;

    echo “Numero di telefono: “. $_POST[‘telefono’]. “<br />”;

    echo “Data di arrivo: “. $_POST[‘gg’]. “/”. $_POST[‘mm’]. “/”. $_POST[‘aaaa’]. “<br />”;

    echo “Numero notti: “. $_POST[‘notti’]. “<br />”;

    echo “Tipo di camera: “. $_POST[‘camera’]. “<br />”;

    echo $_POST[‘colazione’];

  }

}

else

{

  echo ‘Non è stato compilato il form per la prenotazione.’;

}



?>
```

Questo è un form completo di controlli! Abbiamo inserito quindi un grande blocco condizionale che controlla ogni singolo campo del modulo e verifica che i dati inseriti siano corretti.

Concludiamo come al solito con un esercizio. Costruiamo un form per inserire opere d’arte nel nostro sito: un campo ci chiede quante opere vogliamo inserire, e a seconda del numero ci vengono restituiti altrettanti campi “opera”, “autore” e “datazione”. Alla fine compiliamo tutti i dati per ciascuna opera e stampiamo il tutto.

16. Alcune funzioni da non dimenticare! Parte 1: funzioni generali e per le stringhe

Con il capitolo presente, iniziamo una piccola “serie”, di tre capitoli, con la quale scopriremo l’uso di **alcune delle funzioni più importanti di PHP**: stiamo parlando ovviamente di **funzioni predefinite**. In questo capitolo ci dedicheremo ad alcune funzioni che riguardano **le variabili in senso generale** e **le stringhe**. Il prossimo sarà invece dedicata alle funzioni per **gli array** e infine il terzo e ultimo capitolo di questo piccolo ciclo si occuperà delle funzioni per **la scrittura e la lettura su file esterni**. Per tutte queste funzioni (così come per quelle che abbiamo anche visto nelle lezioni scorse) esiste un’**ampia documentazione** sul sito ufficiale del linguaggio PHP, http://php.net: per ogni funzione troverete **sintassi ed esempi d’uso**.

Nel capitolo precedente abbiamo visto una delle più importanti funzioni per le variabili, ovvero la funzione **isset**, che controlla se una variabile **è settata o meno**, e restituisce **true in caso positivo e false in caso negativo**. Ne abbiamo visto la sintassi e abbiamo osservato quanto si rivela utile soprattutto se dobbiamo controllare alcuni form.

Partiamo dalla funzione **unset**: serve per **eliminare una variabile**. La sintassi è molto semplice, ci basterà includere tra le parentesi tonde la variabile che vogliamo eliminare:

```
<?php



$variabile = “Stringa”;

unset($variabile);



?>
```

Se, dopo aver invocato la funzione *unset*, proveremo a stampare la variabile $variabile, **ci sarà un errore** proprio perché abbiamo “distrutto” la variabile tramite la funzione *unset*. In realtà **non si tratta di una vera funzione**, perché non restituisce alcunché: possiamo però chiamarla così perché **nacque inizialmente come una vera funzione** e a partire da PHP 4 è diventata un comando (come *echo* o *include*). *Unset* ci tornerà utile quando ci occuperemo dei cookie e dovremo distruggerli. La pagina dove poter trovare la documentazione ufficiale di *unset* è http://php.net/manual/en/function.unset.php.

La prossima funzione è **is_null**: controlla se il valore associato una variabile **è nullo oppure no**. **In caso affermativo**, la funzione restituirà il booleano **true** e **in caso**

contrario restituirà false. Anche questa funzione riceve come parametro semplicemente **la variabile che vogliamo controllare**. Vediamola applicata in un blocco condizionale:

```
<?php



$variabile = NULL;



if (is_null($variabile))

{

   echo “Variabile nulla”;

}

else

{

   echo $variabile;

}



?>
```

In questo caso la variabile **è nulla** (per dichiarare nulla una variabile basta scrivere, al posto del valore, la parola chiave **NULL, tutta maiuscola**), quindi la funzione **restituirà true**. Se alla variabile avessimo invece abbinato un valore, il blocco condizionale avrebbe stampato la nostra variabile. Questa funzione può essere molto utile quando si opera con i database MySQL: capita infatti spesso che ci siano record vuoti nel database, e per controllarli si può utilizzare questa funzione. La pagina della documentazione ufficiale è http://php.net/manual/en/function.is-null.php.

La prossima funzione, **empty**, controlla **se una variabile è vuota**, ed è molto simile a *is_null*: in caso affermativo, restituirà *true* e in caso negativo *false*. Vediamo ancora la sintassi applicata in un blocco condizionale:

```
<?php
```

```
$variabile = "";

if (empty($variabile))
{
   echo "Variabile vuota";
}
else
{
   echo "La variabile $variabile non è vuota";
}

?>
```

Un esempio pratico dell'utilizzo della funzione empty potrebbe essere questo: controllare se un campo del form è stato compilato oppure no. C'è però una certa differenza tra *is_null* ed *empty*: infatti **una variabile nulla è anche una variabile vuota**, ma **una variabile vuota può anche non essere nulla**. Nel caso dell'esempio la variabile **è vuota ma non è nulla**. La pagina della documentazione ufficiale della funzione *empty* si trova qui: http://php.net/manual/en/function.empty.php.

Passiamo ora alle funzioni che riguardano le stringhe. Ne abbiamo vista una nel precedente capitolo: **strlen**, che controlla **la lunghezza della stringa** e restituisce **il valore corrispondente** (nel caso di una stringa "Ciao", scrivendo echo strlen($Ciao); otterremo come risultato 4). Adesso vediamo la funzione **substr**, che restituisce **una sottostringa partendo da un carattere ben preciso**.

Se per esempio vogliamo estrarre, dalla stringa "Io mi chiamo Federico" la sottostringa "Federico", dovremo invocare la nostra funzione *substr* passando come parametro la stringa, il numero 13, ovvero **il carattere di partenza** della nostra sottostringa, e il numero 8, ovvero **il numero di caratteri che vogliamo considerare**:

```
<?php
```

```
$stringa = “Io mi chiamo Federico”;

echo substr($stringa,13,8);



?>
```

Il risultato della nostra stampa (e della nostra funzione) non sarà altro che la parola “Federico”. Se volessimo poi assegnare la stringa “Federico” come **valore di una nuova variabile** è sufficiente una dichiarazione che associ alla nuova variabile **nient’altro che la funzione**:

```
<?php



$stringa = “Io mi chiamo Federico”;

$nome = substr($stringa,13,8);

echo $nome;



?>
```

Si tratta di una funzione utile se ci troviamo a dover trattare un testo da cui **estrapolare alcune parole**. Possiamo utilizzare poi questa variabile anche con numeri negativi, per esempio se vogliamo **contare la stringa partendo dal fondo**:

```
<?php



$stringa = “Io mi chiamo Federico”;

$nome = substr($stringa,-8,8);

echo $nome;



?>
```

In questo caso significa che vogliamo far partire la stringa dall'ottavo carattere leggendo però la stringa **da destra verso sinistra**. La documentazione ufficiale della funzione *substr* si trova qua: http://php.net/manual/en/function.substr.php.

Una funzione molto utile è *str_replace*, che ci consente di **sostituire sottostringhe all'interno di una stringa**: una eventualità che ci si può presentare in una varietà molteplice di occasioni (sostituire il nome di una persona in un testo, correggere errori, modificare etichette HTML formattate male).

Esempio molto banale: supponiamo che qualcuno ci abbia inviato un testo in HTML con i break scritti male. Per non doverli correggere uno a uno possiamo utilizzare un editor oppure il PHP, con la funzione *str_replace*, che riceve tre parametri. Il primo è **la sottostringa da rimpiazzare**, il secondo è **la nuova sottostringa** e il terzo è **la stringa che contiene il tutto**:

```
<?php



$testo = "Federico<br>Ilaria<br>Gabriele<br>";

$testo = str_replace("<br>","<br />",$testo);

echo $testo;



?>
```

In questo modo, all'interno della stringa $testo (per semplicità io ho inserito una semplice lista di nomi, ma anche un testo molto lungo e articolato è considerato, in PHP, come una stringa, se associato come valore a una variabile), le tag formattate male saranno **correttamente sostituite** con etichette rispettose degli standard internazionali: sarà sostituita ogni occorrenza della sottostringa <br>, dove con occorrenza si intende ogni singola comparsa di quella sottostringa. La documentazione ufficiale è in questa pagina: http://php.net/manual/en/function.str-replace.php.

Abbiamo poi un'altra funzione utile, **substr_count**, che serve per **contare le occorrenze di una sottostringa in una stringa**. Riceve come primo parametro **la stringa** e come secondo **la sottostringa di cui vogliamo conoscere il numero di occorrenze**, che viene restituito come risultato. In questo esempio:

```
<?php



$testo = “Federico<br />Ilaria<br />Gabriele<br />”;

echo substr_count($testo,“Federico”);



?>
```

Il risultato della funzione sarà 1. Attenzione: la funzione è **case sensitive**, quindi scrivere “Federico” e scrivere “federico” non è la stessa cosa. Ecco qua la documentazione ufficiale: http://php.net/manual/en/function.substr-count.php.

Infine due funzioni molto simili, per trasformare i caratteri di una stringa: nel primo caso, con la funzione **strtolower**, faremo diventare tutti minuscoli i caratteri della nostra stringa e nel secondo caso, con la funzione **strtoupper**, tutti maiuscoli. Entrambe ricevono come parametro semplicemente **la variabile da trasformare**:

```
<?php



$stringa = “Federico”;

echo strtoupper($stringa);

echo “<br />”;

echo strtolower($stringa);



?>
```

Nel primo caso la funzione trasformerà la stringa in “FEDERICO”, nel secondo caso la trasformerà in “federico”. La documentazione ufficiale di *strtoupper* è qui http://php.net/manual/en/function.strtoupper.php mentre quella di *strtolower* invece è qui http://php.net/manual/en/function.strtolower.php.

Per la gestione delle stringhe ci sono moltissime funzioni, queste che abbiamo visto sono

le principali ma potete consultare il sito php.net alla ricerca di altre funzioni, oppure potete acquistare il libro **“PHP” di Rasmus Lerdorf**, dove troverete tutte le funzioni di PHP **divise a seconda dell’argomento** (funzioni per gli array, per i database, per le stringhe ecc. ecc.). È edito in Italia da Hops e si trova facilmente in tutte le librerie rifornite in campo informatico. È un libretto un po’ vecchiotto ma è utile per avere a portata di mano tutte le funzioni!

Terminiamo con un esercizio. Create un form per cercare il numero di occorrenze di una forma all’interno della vostra poesia preferita. Non devono esserci differenze tra forme minuscole e maiuscole quindi se io cerco “ciao” voglio che mi vengano contate assieme le occorrenze di “ciao” scritto minuscolo ed eventuali “Ciao” scritti con la lettera maiuscola. Mi spiego meglio: se in un testo compaiono una volta la forma “ciao” e un’altra volta la forma “Ciao”, voglio che il risultato sia 2 e non 1, anche perché “ciao” e “Ciao” sono la stessa forma (c’è solo una differenza grafica nel primo carattere). La soluzione, come sempre, in fondo all’ebook.

17. Alcune funzioni da non dimenticare! Parte 2: gli array

Eccoci arrivati al secondo appuntamento con la nostra panoramica sulle funzioni più utili di PHP. Nel capitolo precedente abbiamo visto alcune funzioni che hanno validità generale oltre ad alcune funzioni per le stringhe: in questo ne vediamo quattro con le quali possiamo fare **operazioni con gli array**.

In realtà conosciamo già **una funzione per gli array**, e l'abbiamo vista nel capitolo sui controlli iterativi: si tratta della funzione **count** che, come ricorderete, ci serve per **contare quanti elementi sono contenuti in un array**. In questo capitolo cominciamo invece con un paio di funzioni che ci servono per **trasformare una stringa in un array** e per **fare l'operazione inversa**: hanno dei nomi facili da ricordare e si chiamano, rispettivamente, **explode** e **implode**.

Cominciamo con **explode**, la funzione che **trasforma una stringa in un array**. Poniamo per esempio di avere una variabile $stringa a cui associamo un elenco di nomi: "Federico, Ilaria, Gabriele". La funzione *explode* deve ricevere **due stringhe come parametri**: la prima sarà il **"separatore"**, ovvero l'elemento che dovrà far capire alla funzione **dove finisce un elemento** del nuovo array che si andrà a creare e dove invece **avrà inizio il successivo**, e la seconda invece sarà, ovviamente, **la stringa che dovremo trasformare in un array**. Invochiamo quindi la nostra funzione in questo modo:

```
$stringa = "Federico, Ilaria, Gabriele";

$array = explode(", ", $stringa);
```

Adesso abbiamo un array che possiamo stampare con i modi che già conosciamo:

```
echo $array[0]; // stampiamo il primo elemento

foreach ($array as $elemento)

{ // stampiamo tutti gli elementi

   echo "$elemento<br />";

}
```

Abbiamo quindi scelto, come stringa per la separazione, la virgola seguita dallo spazio. Possiamo anche specificare **un numero intero dopo il secondo parametro**: significa che

l’array dovrà avere **il numero di elementi specificato**, per cui, raggiunto il limite, l’ultimo elemento sarà tutto ciò che nella stringa si trova **dopo il separatore corrispondente al numero indicato**. Ma sono convinto che un esempio possa chiarire meglio. Se scriviamo una cosa del genere:

```
$stringa = “Federico, Ilaria, Gabriele”;

$array = explode(“, “, $stringa, 2);
```

Avremo un array di due elementi: al primo sarà associato il valore “Federico” e al secondo il valore “Ilaria, Gabriele”. Possiamo anche specificare **un numero negativo**: così facendo faremo in modo che dal nostro array vengano esclusi **gli ultimi n elementi** che specifichiamo con il nostro intero negativo. Per esempio:

$stringa = “Federico, Ilaria, Gabriele”;

$array = explode(“, “, $stringa, -1);

In questo modo, scrivendo -1, sarà escluso l’ultimo elemento (“Gabriele”) e l’array risulterà composto dai primi due elementi. Se avessimo scritto -2 avremmo escluso “Gabriele” e “Ilaria”, e così via. La documentazione ufficiale della funzione *explode* si trova alla pagina http://php.net/manual/en/function.explode.php.

Abbiamo poi la funzione che ci permette, come anticipato prima, di fare l’operazione inversa, ovvero **trasformare un array in una stringa**. Il “meccanismo” è simile a *explode*: la funzione riceve due parametri, con il primo che è una stringa che ricoprirà la funzione di **separatore degli elementi dell’array** e il secondo parametro che è, ovviamente, **l’array**. Prendiamo il nostro solito array di nomi e vediamo come comportarci:

```
$array = array(“Federico”, “Ilaria”, “Gabriele”);

$stringa = implode(“, “, $array);
```

Con la funzione *implode* abbiamo quindi specificato di creare una stringa contenente i nomi memorizzati nel nostro array, separati da una virgola e da uno spazio, cosicché il valore associato alla variabile $stringa sarà “Federico, Ilaria, Gabriele”.

Possiamo anche **omettere il primo parametro**:

```
$array = array(“Federico”, “Ilaria”, “Gabriele”);

$stringa = implode($array);
```

E in questo caso avremo una stringa con i nomi tutti attaccati: “FedericoIlariaGabriele”. Purtroppo la funzione implode non prevede, come *explode*, che vengano passati terzi parametri, quindi se vogliamo escludere elementi dall’array dovremo arrangiarci un po’ a seconda di ciò che vogliamo fare del nostro array. Trovate la documentazione ufficiale della funzione implode alla pagina http://php.net/manual/en/function.implode.php.

Veniamo ora alla funzione **sort**, utile per **ordinare gli elementi di un array in ordine alfabetico**. Riceve come parametro la variabile che contiene il nostro array:

```
$array = array(“Federico”, “Ilaria”, “Gabriele”);

sort($array);

foreach ($array as $elemento)

{

   echo “$elemento<br />”;

}
```

In seguito alla stampa, noteremo proprio che gli elementi saranno **ordinati secondo la loro successione alfabetica**. La funzione *sort* restituisce un valore booleano e la sua pagina di documentazione ufficiale è questa: http://php.net/manual/en/function.sort.php. Attenzione! Dopo il *sort*, gli elementi dell’array **avranno i loro indici cambiati**. Per esempio, $array[1] non sarà più Ilaria ma sarà Gabriele. Esiste tuttavia una funzione che ci permette di **disporre gli elementi in ordine alfabetico senza alterarne gli indici**, ed è la funzione **asort**. Si comporta **esattamente come sort**:

```
$array = array(“Federico”, “Ilaria”, “Gabriele”);

asort($array);

foreach ($array as $elemento)

{
```

```
  echo “$elemento<br />”;

}
```

Con la differenza che $array[1] sarà sempre “Ilaria”. La documentazione ufficiale di *asort* si trova qui: http://php.net/manual/en/function.asort.php.

Esistono poi le due funzioni **corrispettive di sort e asort che svolgono l’operazione contraria**, ovvero dispongono gli elementi dell’array **secondo il loro ordine alfabetico decrescente**, e sono le funzioni **rsort e arsort**.

```
$array = array(“Federico”, “Ilaria”, “Gabriele”);

rsort($array);

//oppure arsort($array);

foreach ($array as $elemento)

{

  echo “$elemento<br />”;

}
```

La documentazione ufficiale di queste funzioni si trova, per *rsort*, alla pagina http://php.net/manual/en/function.rsort.php e, per *arsort*, alla pagina http://php.net/manual/en/function.arsort.php.

Un’altra funzione utile è **in_array**: ci permette di cercare un elemento nell’array e restituisce il booleano **true** nel caso in cui l’elemento **venga trovato** e **false nel caso contrario**. Riceve come primo parametro **il valore dell’elemento che vogliamo cercare** e come secondo parametro **l’array in cui lo vogliamo cercare**:

```
$array = array(“Federico”, “Ilaria”, “Gabriele”);

if (in_array(“Federico”, $array))

{

  echo “Ho trovato Federico!”;
```

```
}

else
{

   echo “Non ho trovato Federico.”;

}
```

In questo caso la condizione sarà soddisfatta perché “Federico” è presente nell’array $array, quindi il nostro blocco stamperà la scritta “Ho trovato Federico!”. La documentazione ufficiale si trova qui: http://php.net/manual/en/function.in-array.php. Esiste anche la funzione per **cercare un indice** e non un valore, si chiama **array_key_exists** e funziona proprio come *in_array*:

```
$array = array(“Federico”, “Ilaria”, “Gabriele”);

if (array_key_exists(“2”, $array))

{

   echo “L’indice numero 2 esiste!”;

}

else

{

   echo “Non esiste l’indice numero 2.”;

}
```

L’indice numero 2 corrisponde all’elemento “Gabriele”, perciò la funzione restituirà “true” e la condizione del blocco condizionale sarà soddisfatta. Ecco qua la documentazione ufficiale: http://php.net/manual/en/function.array-key-exists.php.

Chiudiamo infine con due funzioni molto utili: una serve per **aggiungere elementi a un array**, e si chiama **array_push**, e l’altra serve per **fondere due o più array insieme** e si chiama **array_merge**.

La funzione *array_push* riceve come primo parametro **l’array a cui vogliamo aggiungere elementi**, e poi, come parametri successivi *ad libitum*, **tutti gli elementi che vogliamo**

inserire nell'array:

```
$array = array("Federico", "Ilaria");

array_push($array, "Gabriele");

foreach ($array as $elemento)

{

   echo "$elemento<br />";

}
```

Abbiamo iniziato con un array di due elementi, "Federico" e "Ilaria", e ci abbiamo aggiunto l'elemento "Gabriele". Una funzione molto importante e molto utile quindi, che ci consente di inserire tanti elementi aggiuntivi a un array tutti in una volta sola. La documentazione ufficiale si trova qui: http://php.net/manual/en/function.array-push.php.

L'ultima funzione che vediamo in questo capitolo è *array_merge* e, come anticipato, ci serve per **unire due o più array**. Riceve come parametri **tutti gli array che vogliamo fondere assieme**:

```
$array1 = array("Federico", "Ilaria");

$array2 = array("Gabriele");

$array = array_merge($array1, $array2);
```

Abbiamo quindi creato un nuovo array $array risultante dalla fusione tra $array1 e $array2. Gl indici degli elementi del secondo array saranno accodati a quelli del primo, quelli del terzo a quelle del secondo e così via. **Fate particolare attenzione agli array associativi**. Può capitare infatti di avere due array che abbiano **due indici uguali**:

```
$array1 = array("a" => "Federico", "b" => "Ilaria");

$array2 = array("b" => "Gabriele");

$array = array_merge($array1, $array2);
```

Se decidiamo di fondere i due array, come abbiamo fatto in questo esempio, la variabile che ha un certo indice **sovrascriverà l'elemento con lo stesso indice presente nell'array precedente**. Quindi in questo caso "Gabriele" sovrascrive "Ilaria", che sarà eliminato dall'array. Funziona un po' a "cascata", perché se avessimo avuto un terzo array con un nuovo elemento abbinato a un indice "b", quest'ultimo avrebbe sovrascritto sia "Ilaria" che "Gabriele". La documentazione ufficiale di array_merge si trova alla pagina http://php.net/manual/en/function.array-merge.php.

Ma eccovi l'esercizio! Dati tre array:

```
$array1 = array("a", "b", "c");

$array2 = array("d", "e", "f");

$array3 = array("g", "h", "i");
```

Uniteli in un unico array, escludete gli ultimi due elementi quindi uniteli in una stringa e stampate il tutto.

18. Alcune funzioni da non dimenticare! Parte 3: gestire file di testo esterni

Eccoci arrivati al terzo e ultimo capitolo con le nostre funzioni da non dimenticare, ed è un capitolo importante perché vedremo le funzioni che ci servono **per la gestione di file di testo esterni**. Con PHP è infatti possibile creare e scrivere file "al momento", e non è neanche troppo difficile: basta fare un po' di attenzione.

Innanzitutto dobbiamo **settare i permessi della cartella in cui scriveremo i file** in modo tale da **renderla scrivibile**: se l'abbiamo impostata in sola lettura il nostro codice non funzionerà. La prima cosa da fare per gestire un file esterno **è crearlo**: la funzione che fa al caso nostro è **fopen** (che ovviamente sta per "file open"), che in realtà serve per **aprire un file**, ma se il file che specifichiamo come parametro **non esiste**, **la funzione lo crea**. Riceve due parametri: il primo è **una stringa contenente il nome (e, di conseguenza, il percorso) del file**, e la seconda è la **"modalità"**. Chiariamo meglio con un esempio:

```
$file = fopen('file.txt', 'w');
```

In questo caso abbiamo aperto, nella cartella della pagina che stiamo eseguendo, il file chiamato "file.txt", e in particolare lo abbiamo aperto **in modalità "scrittura"** ("w" che sta per "write"). Se il file non è presente, viene creato in automatico. Vediamo quali sono le varie modalità:

- r: file aperto in sola lettura;
- r+: file aperto in lettura e scrittura (se non esiste NON viene creato);
- w: file aperto in sola scrittura;
- w+: file aperto in lettura e scrittura (se non esiste viene creato);
- a: file aperto in sola scrittura e i contenuti vengono aggiunti (con w invece vengono sovrascritti a quelli eventualmente presenti in precedenza. Se il file non esiste viene creato);
- a+: file aperto in lettura e scrittura e i contenuti vengono aggiunti (se il file non esiste viene creato);
- x: file creato in scrittura (se esiste la pagina restituirà un errore);
- x+: file creato in lettura e scrittura (se esiste la pagina restituirà un errore);
- c: file aperto in scrittura, i contenuti vecchi non sono cancellati e quelli nuovi sono aggiunti all'inizio sovrascrivendo i contenuti vecchi fino al punto in cui arrivano i contenuti nuovi;
- c+: come c, in lettura e scrittura.

Abbiamo quindi **molte opportunità per gestire la scrittura dei contenuti sul file** (ora vedremo come si fa), e la scelta dipende da quello che dobbiamo fare del nostro file. Se **non ci interessa salvare i contenuti** sarà sufficiente **la modalità w**, se invece **ci interessa aggiungere ogni volta contenuti nuovi** dovremo usare la **modalità a**.

Possiamo specificare il percorso del nostro file:

```
$file = fopen('../file.txt', 'w');

$file = fopen('file-esterni/file.txt', 'w');
```

Però in quest'ultimo caso **la cartella deve esistere**: la sola *fopen* non è sufficiente per creare la cartella (a proposito, la documentazione ufficiale di *fopen* è qui: http://php.net/manual/en/function.fopen.php).

Per creare una directory ci servirà la funzione **mkdir**, che riceve come parametro **il nome della cartella che vogliamo creare**:

```
mkdir('file-esterni');
```

La documentazione ufficiale di *mkdir* è alla pagina http://php.net/manual/en/function.mkdir.php. **Attenzione però**: nel caso in cui la cartella sia già esistente e voi invochiate comunque la funzione *mkdir*, **ci sarà un warning** che vi farà notare **l'esistenza della cartella**. Per evitare questo, si utilizza tipicamente la funzione **is_dir**, che serve per **verificare se una cartella esiste già** e riceve come parametro **la stringa con il nome della cartella**. Eccovi un tipico esempio:

```
if(!is_dir('file-esterni'))

{ // se la cartella "file-esterni" non esiste…

   mkdir('file-esterni'); // … allora la creiamo…

}

else

{

//… altrimenti non facciamo niente.
```

```
}
```

Con questo blocco if diciamo al preprocessore che se la cartella “file-esterni” non esiste (il punto esclamativo davanti alla funzione *is_dir*, come ricorderete, ha valore negativo) andrà creata tramite la funzione *mkdir*. Se invece esiste già si passa all’else che non contiene istruzioni, quindi non facciamo niente. Scrivere !is_dir(‘file-esterni’) equivale a scrivere is_dir(‘file-esterni’) == false, perché la funzione restituisce un valore booleano. La documentazione ufficiale di *is_dir* si può trovare qua: http://www.php.net/manual/en/function.is-dir.php.

Ora che abbiamo visto come creare un file (insieme all’eventuale cartella) vediamo come riempirlo! E per fare ciò utilizziamo la funzione **fwrite**, che riceve come parametri **il file su cui vogliamo scrivere e la stringa da scrivere nel file**:

```
$file = fopen(‘file.txt’,‘a’);

fwrite($file, “Ciao, io mi chiamo <strong>Federico</strong>!”);
```

Una funzione quindi molto semplice da usare. Abbinandola in questo caso alla funzione *fopen* in **modalità add (“a”)**, abbiamo aggiunto la stringa specificata dopo gli eventuali contenuti nel file. Se avessimo invocato la funzione *fopen* in modalità **write**, la nostra stringa **avrebbe cancellato tutti i contenuti del file file.txt**. La documentazione ufficiale di *fwrite* è in questa pagina: http://php.net/manual/en/function.fwrite.php.

Per quanto riguarda la lettura dei file, ci sono due funzioni che fanno al caso nostro: **file_get_contents** e **fread**. Partiamo dalla prima, che è più semplice: riceve come parametro **il nome del file di testo** e **converte il suo contenuto in una stringa**. Si usa così:

```
<?php



$stringa = file_get_contents(‘file.txt’);

echo $stringa;



?>
```

Molto semplice, non serve neppure aprire il file con la funzione **fopen**. La documentazione ufficiale è qui: http://php.net/manual/en/function.file-get-contents.php.

La funzione **fread** è un pochino più articolata, e la differenza rispetto a *file_get_contents* consiste nel fatto che ci dà la possibilità di leggere anche solo **una porzione di file** (invece *file_get_contents* lavora **sull'intero contenuto del file**). Per usare *fread* poi, a differenza di *file_get_contents*, è necessario anche aprire il file con fopen. La funzione riceve come parametri **il file aperto con fopen** e **il numero di byte che vogliamo leggere**:

```
<?php



$file = fopen('file.txt', 'r');

$stringa = fread($file, 20);

echo $stringa;



?>
```

Nel caso di questo esempio leggiamo **i primi 20 byte del nostro file**. La documentazione di *fread* si trova qui: http://it.php.net/manual/en/function.fread.php.

E ovviamente, dopo aver eseguito tutte le nostre operazioni, è buona prassi **chiudere il file esterno**. Lo facciamo con la funzione **fclose**, che molto semplicemente riceve come parametro **il file da chiudere**:

```
fclose('file.txt');
```

Perché si chiude un file? Perché se succede qualcosa al sito mentre noi o i nostri utenti stanno compiendo operazioni e il file non è chiuso, questo **potrebbe danneggiarsi**, quindi **chiudiamo sempre i file dopo che li abbiamo aperti**. Questa funzione restituisce **un valore booleano** (ci serve se vogliamo controllare, magari, se un file è stato chiuso bene) e la sua documentazione ufficiale è alla pagina http://it.php.net/manual/en/function.fclose.php. È anche buona prassi, nel caso in cui dobbiamo leggere con **fread** un file precedentemente scritto con **fwrite**, chiuderlo e poi riaprirlo in modalità sola lettura.

19. Costruiamo un guestbook con PHP

Il presente capitolo costituisce un'**applicazione pratica** di tutto quello che abbiamo visto finora: controlli, funzioni, gestione di file di testo esterni e quant'altro. Impariamo quindi a realizzare **un pratico ed efficace guestbook** per il nostro sito avvalendoci soltanto di **PHP e HTML** senza passare da **database**.

Abbiamo bisogno di quattro file per far funzionare il nostro guestbook: una pagina PHP per l'**inserimento del messaggio** da parte degli utenti, la pagina PHP dove **saranno mostrati i messaggi**, un **file di testo** dove i messaggi **saranno memorizzati** e **un ulteriore file di testo** in cui, per sicurezza, **memorizzeremo gli indirizzi IP** dei visitatori che scrivono sul sito. **Attenzione**: il file con gli IP dovrà essere protetto in modo tale da **non permettere a nessuno di poter leggere il file** (sapete com'è… questioni di privacy). Vi anticipo già quale sarà l'**esercizio finale** del capitolo: il guestbook che adesso creiamo avrà i messaggi ordinati in modo tale che i primi siano quelli più vecchi e gli ultimi siano quelli più recenti. Come esercizio vi propongo di modificare il guestbook che vedremo in questo capitolo per far sì che **i messaggi più recenti siano in testa**.

Ma veniamo subito al guestbook. Creiamo i nostri due file PHP, che chiameremo guestbook.php e messaggio.php. Il primo file su cui operiamo è messaggio.php, e la prima cosa che dobbiamo fare è **creare due funzioni** per **mostrare il form** all'utente. Perché due funzioni? La prima servirà per mostrare all'utente il form quando **arriva per la prima volta** nella pagina messaggio.php. Ma noi controlleremo se il modulo è stato compilato bene, e **in caso di errore avviseremo il nostro utente**. Cosa potrebbe succedere? Potrebbe succedere che in caso di errore la pagina si ricarichi **perdendo i dati inseriti**. È **molto antipatico** un guestbook in cui un utente scrive **un messaggio lungo** e poi per un errore (magari ha dimenticato di inserire la mail) il messaggio **gli viene cancellato e deve riscriverlo da capo**. Quindi la seconda funzione ci servirà per **tenere memoria del post dell'utente**. Sono due funzioni molto semplici. Vediamo la prima, che chiameremo semplicemente "mostraForm":

```
function mostraForm()

{

    echo "

   <form method='post'>

   Inserisci il tuo nome: <br />

   <input type='text' size='50' name='utente' /><br />
```

```
  Inserisci la tua e-mail (non sarà pubblicata): <br />

  <input type=‘text’ size=‘50’ name=‘email’ /><br />

  Inserisci il messaggio:<br />

  <textarea name=‘messaggio’ rows=‘10’ cols=‘50’></textarea>

  <br /><br />

  <input type=‘submit’ name=‘submit’ value=‘Firma’>

  <input type=‘reset’ valur=‘Cancella’>

  </form>”;

}
```

Come vedete non c’è niente di strano: un semplice *echo* che stampa un form con tre campi, due *input* (uno per il nome e uno per la mail) e una *textarea* per il messaggio, oltre a due pulsanti (uno per cancellare tutto e uno per inviare il messaggio). Vediamo ora la funzione che **tiene conto dei dati inseriti dall’utente** in caso di errore, e che chiamiamo “mostraFormConDati”:

```
function mostraFormConDati()

{

  echo “

  <form method=‘post’>

  Inserisci il tuo nome:<br />

  <input type=‘text’ size=‘50’ name=‘utente’ value="”; echo stripslashes($_POST[‘utente’]); echo “" />
<br />

  Inserisci la tua e-mail (non sarà pubblicata): <br />

  <input type=‘text’ size=‘50’ name=‘email’ value=’”; echo $_POST[‘email’]; echo ”’ /><br />

  Inserisci il messaggio: <br />

  <textarea name=‘messaggio’ rows=‘10’ cols=‘50’>”; echo stripslashes($_POST[‘messaggio’]); echo
”</textarea>
```

```
  <br /><br />

  <input type=‘submit’ name=‘submit’ value=‘Firma’>

  <input type=‘reset’ valur=‘Cancella’>

  </form>”;

}
```

Vedete? Come attributo *value* dei campi “utente” ed “email” (e come testo della textarea “messaggio) abbiamo inserito i dati del nostro modulo (tramite metodo post), passati però attraverso la funzione **stripslashes**. Bene quindi impariamo una funzione nuova! A cosa serve la funzione *stripslashes*? Ricorderete che in uno dei primi capitoli abbiamo detto che in alcuni casi **gli apostrofi possono essere scambiati per apici**, e in questo caso, per far capire al preprocessore che si tratta proprio **di un apostrofo e non di un apice**, bisogna utilizzare il carattere di **escape**.

Se ci avete fatto caso lo abbiamo utilizzato anche nell’attributo *value* dell’input “utente”: questo perché, se un utente inserisce un nome con l’apostrofo, **potrebbero esserci dei problemi** in quanto, come ben sapete, in HTML l’apostrofo del nome verrebbe interpretato come **fine dell’attributo value**. Ma torniamo alla nostra *stripslashes*, che è una delle funzioni più importanti di PHP. Ci permette di **eliminare i caratteri di escape che vengono inseriti** (o da noi manualmente o tramite un modulo come in questo caso), e riceve come parametro semplicemente **la stringa dalla quale rimuovere i caratteri di escape**. Questo perché non è bello vedere una frase piena di backslash. La documentazione ufficiale si trova in questa pagina: http://php.net/manual/en/function.stripslashes.php.

A questo punto, dichiarate le nostre due funzioni, facciamo partire **un grande blocco condizionale** che controlla se l’utente **ha premuto il pulsante “invia”** (lo facciamo con la funzione *isset*): in caso negativo, viene invocata la funzione “mostraForm”:

```
if (!isset($_POST[‘submit’]))

{

  showForm();

}
```

In caso affermativo invece apriamo **un grande else** che conterrà **tutto il resto della**

gestione del messaggio dell’utente:

```
else

{

   // operazioni, operazioni…

}
```

Vediamo come riempire questo *else*. La prima cosa da fare è dichiarare **una variabile numerica**, che chiameremo “errore” e che **imposteremo a zero**. Perché questo? Perché in caso di **errata compilazione** del modulo mostreremo all’utente **dove ha sbagliato**, e ci servirà una variabile per controllare **quali campi sono stati compilati in modo errato dall’utente**. Per cui, per ogni errore, imposteremo la variabile $errore a 1 e dichiareremo **un elemento di un array di stringhe**: questo elemento conterrà **la frase da mostrare all’utente per indicargli dove ha sbagliato**. Ma vediamo il codice per comprendere meglio:

```
$errore = 0;

if(empty($_POST[‘utente’]))

{

   $errore = 1;

   $errstr[] = “Inserisci il tuo nome…”;

}

if (empty($_POST[‘messaggio’]))

{

   $errore = 1;

   $errstr[] = “Inserisci il tuo messaggio…”;

}

if (empty($_POST[‘email’]))

{

   $errore = 1;
```

```
  $errstr[] = “Inserisci la tua email…”;

}
```

Quindi, se questi tre campi **risultano vuoti** (controlliamo tramite la funzione **empty**), come detto impostiamo la variabile $errore a 1 e dichiariamo un elemento di un array, che chiameremo $errstr (stringhe di errore). E adesso è arrivato il momento di **mostrare all’utente i suoi errori**! Lo facciamo semplicemente controllando la variabile $errore. Se è uguale a 1 significa che c’è stato qualcosa che non è andato, quindi, con un *foreach*, stampiamo gli elementi dell’array $errstr per far vedere all’utente dove ha sbagliato e quindi, terminato il *foreach*, gli ristampiamo il form con i dati memorizzati, per evitargli l’**antipaticissimo compito di riscrivere il messaggio**:

```
if ($errore == 1)

{

  foreach($errstr as $err)

  {

    echo “<li><strong> ” , $err , “</strong></li>”;

  }

  echo “<br />”;

  showFormEnhanced();

}

else

{

  // se tutto va bene…

}
```

A questo punto, se è andato tutto bene, facciamo partire **l’else riferito a quest’ultimo if**, quindi vediamo cosa succede **se la variabile $errore rimane uguale a zero** (ovvero se le operazioni sono andate a buon fine). Per prima cosa dichiariamo tre variabili, $utente, $messaggio ed $email, a cui assoceremo il valore dei campi del modulo fatti passare attraverso *stripslashes*:

```
$utente = stripslashes($_POST[‘utente’]);

$messaggio = stripslashes($_POST[‘messaggio’]);

$email = stripslashes($_POST[‘email’]);
```

Una cosa molto carina è memorizzare **la data e l’orario del messaggio**. Per fare questo utilizziamo un’altra funzione di PHP, **date**, che serve proprio **per stampare date**. Riceve come parametro **il formato della data che più ci piace**:

```
$data_ora = date(“d/m/Y H:i:s”);
```

Noi abbiamo scelto un formato data/mese/anno ore:minuti:secondi, ma ce ne sono tanti altri e potete utilizzare quelli che più vi piacciono. Qualche esempio:

- d-m-Y
- d.m.Y
- se siete anglosassoni anche m d Y
- ecc. ecc.

Basta solo rispettare le lettere indicate: **d** per il giorno, **m** per il mese, **Y** (maiuscolo) per l’anno, **H** (maiuscolo) per l’ora, **i** per il minuto e **s** per il secondo. E poi ci sono **tante altre modalità**: per esempio, se aggiungete **l** vi viene scritto anche il giorno della settimana, se scrivete **F** (maiuscolo) il mese viene scritto in lettere invece che in numeri e così via. Per trovare **tutte le opzioni**, che non vi scrivo perché **sono moltissime**, potete consultare **la documentazione ufficiale** della funzione alla pagina http://it.php.net/manual/en/function.date.php.

Bene, a questo punto creiamo due variabili a cui associamo, come valori, i nostri file di testo esterni:

```
$guestbook_testo = “guestbook.txt”;

$guestbook_testoip = “guestbookIP.txt”;
```

È arrivato ora il momento di **memorizzare il messaggio sui due file esterni**. Ci servirà ciò che abbiamo imparato **nell’ultimo capitolo**. Quindi per prima cosa dovremo aprire il primo file in modalità *add*, per fare in modo che i messaggi vengano inseriti **uno dopo**

l'altro, quindi **memorizziamo il messaggio nel file** e alla fine **chiudiamo il file**. **Ripetiamo** poi la stessa operazione per il file esterno che dovrà **memorizzare gli indirizzi IP**. A proposito, come si fa per ricavare l'indirizzo IP di un utente? Con la variabile superglobale **$_SERVER**, e attraverso l'indice di quest'ultima **REMOTE_ADDR** (tutto maiuscolo):

```
$indirizzoip = $_SERVER['REMOTE_ADDR'];
```

Abbiamo quindi associato l'indirizzo IP dell'utente che scrive il messaggio alla variabile $indirizzoip. Bene, adesso possiamo vedere il codice:

```
$guestbook = fopen("$guestbook_testo","a");

fwrite($guestbook,"<tr><td><b>$utente</b> scrive: <br /><br />

$messaggio<br /><br /><br />

<i>Messaggio scritto in data: $data_ora</i></td></tr>

");

fclose($guestbook);

$ip = fopen("$guestbook_testoip","a");

fwrite($ip,"<tr><td><b><a href='mailto:$email'>$utente</a></b> scrive: <br /><br />

$messaggio<br /><br /><br />

<b>Indirizzo IP utente</b>: $indirizzoip<br /><br />

<i>Messaggio scritto in data: $data_ora</i></td></tr>

");

fclose($ip);
```

Abbiamo memorizzato il messaggio sotto forma di tabella, per renderlo più carino quando lo mostreremo nella pagina guestbook.php. Mi raccomando: includete la stringa che passate come secondo parametro alla funzione *fwrite* **tra doppi apici**, perché vedete che utilizziamo le variabili **senza concatenarle**.

Bene, abbiamo finito il nostro lavoro! **Ringraziamo l'utente** per aver inserito il messaggio:

```
echo “Grazie per aver lasciato un messaggio!<br /><a href=‘guestbook.php’>Torna alla pagina principale del guestbook.</a>”;
```

La pagina guestbook.php è molto più semplice. Inseriamo **un titolo**, **le etichette “table”** perché come detto abbiamo memorizzato i messaggi sotto forma di tabella e **un link alla pagina messaggio.php** e **includiamo il file guestbook.txt**:

```
<h1>Benvenuti nel guestbook!</h1>

<?php


echo “<table>”;

include(“guestbook.txt”);

echo “</table>”;


?>

<br /><br />

<a href=‘messaggio.php’>Firma il guestbook!</a>
```

Ovviamente potete dare alla tabella lo stile che più vi piace, colorare i bordi, insomma con il CSS vi potete divertire come meglio credete.

Il capitolo è concluso. Come vi ho anticipato l’esercizio consiste in questo: fare in modo che i messaggi appaiano nell’ordine dal più recente al più vecchio. I più “temerari” possono anche provare **a suddividere il guestbook in più pagine da 10-20 messaggi l’una**!

Ecco intanto il riepilogo della pagina messaggio.php:

```
<?php


function mostraForm()
```

```
{

   echo “

   <form method=‘post’>

   Inserisci il tuo nome: <br />

   <input type=‘text’ size=‘50’ name=‘utente’ /><br />

   Inserisci la tua e-mail (non sarà pubblicata): <br />

   <input type=‘text’ size=‘50’ name=‘email’ /><br />

   Inserisci il messaggio:<br />

   <textarea name=‘messaggio’ rows=‘10’ cols=‘50’></textarea>

   <br /><br />

   <input type=‘submit’ name=‘submit’ value=‘Firma’>

   <input type=‘reset’ valur=‘Cancella’>

   </form>”;

}

function mostraFormConDati()

{

   echo “

   <form method=‘post’>

   Inserisci il tuo nome:<br />

   <input type=‘text’ size=‘50’ name=‘utente’ value="”; echo stripslashes($_POST[‘utente’]); echo “" />
<br />

   Inserisci la tua e-mail (non sarà pubblicata): <br />

   <input type=‘text’ size=‘50’ name=‘email’ value=’”; echo stripslashes($_POST[‘email’]); echo ”’ /><br
/>

   Inserisci il messaggio: <br />

   <textarea name=‘messaggio’ rows=‘10’ cols=‘50’>”; echo stripslashes($_POST[‘messaggio’]); echo
”</textarea>
```

```
  <br /><br />

  <input type=‘submit’ name=‘submit’ value=‘Firma’>

  <input type=‘reset’ valur=‘Cancella’>

  </form>”;

}

if (!isset($_POST[‘submit’]))

{

  mostraForm();

}

else

{

  $errore = 0;

  if(empty($_POST[‘utente’]))

  {

    $errore = 1;

    $errstr[] = “Inserisci il tuo nome…”;

  }

  if (empty($_POST[‘messaggio’]))

  {

    $errore = 1;

    $errstr[] = “Inserisci il tuo messaggio…”;

  }

  if (empty($_POST[‘email’]))

  {
```

```
    $errore = 1;

    $errstr[] = “Inserisci la tua email…”;
}
if ($errore == 1)
{
    foreach($errstr as $err) {
      echo “<li><strong> ” , $err , “</strong></li>”;
    }
    echo “<br />”;
    mostraFormConDati();
}
else
{
    $utente=stripslashes($_POST[‘utente’]);
    $messaggio=stripslashes($_POST[‘messaggio’]);
    $email=stripslashes($_POST[‘email’]);
    $data_ora = date(“d m Y, l - H:i:s”);
    $guestbook_testo=“guestbook.txt”;
    $guestbook_testoip=“guestbookIP.txt”;
    $indirizzoip = $_SERVER[‘REMOTE_ADDR’];
    $guestbook = fopen(“$guestbook_testo”,“a”);
    fwrite($guestbook,”<tr><td><b>$utente</b> scrive: <br /><br />
```

```
    $messaggio<br /><br /><br />

    <i>Messaggio scritto in data: $data_ora</i></td></tr>

    “);

    fclose($guestbook);

    $ip = fopen(“$guestbook_testoip”,“a”);

    fwrite($ip,”<tr><td><b><a href=‘mailto:$email’>$utente</a></b> scrive: <br /><br />

    $messaggio<br /><br /><br />

    <b>Indirizzo IP utente</b>: $indirizzoip<br /><br />

    <i>Messaggio scritto in data: $data_ora</i></td></tr>

    “);

    fclose($ip);

    echo “Grazie per aver lasciato un messaggio!<br /><a href=‘guestbook.php’>Torna alla pagina principale del guestbook.</a>”;

  }

}



?>
```

Come accennato in apertura di capitolo, l’esercizio consiste nel modificare lo script in modo che nei file di testo, i contenuti più recenti siano in testa.

20. PHP e MySQL

Cominciamo a partire da questo capitolo a conoscere le interazioni **tra PHP e le basi di dati relazionali**. A dire il vero, per esplorare bene l'argomento ci vorrebbe un altro ebook che lo tratti in modo dettagliato, ma quello che impareremo con il nostro corso sarà **sufficiente** per costruire e gestire un sito abbastanza complesso in **PHP-MySQL**.

Questo è un capitolo "introduttivo" all'argomento, quindi in questo capitolo non faremo pratica ma è importante avere qualche **base teorica** per comprendere meglio ciò che affronteremo **successivamente**. Sì, perché prima di costruire un sito con un database **MySQL** è necessario capire **che cos'è un database** e soprattutto **a cosa serve**.

Partiamo dalla prima domanda: che cos'è un **database**? Diciamo innanzitutto che non lo si può chiamare solo "database": possiamo chiamarlo anche **"banca dati"** oppure, ancora meglio, **"base di dati"**, a cui possiamo aggiungere il termine **"relazionale"**.

Diciamo poi che un database è un metodo per **rappresentare la conoscenza**. Detta così è un po' astratta, quindi dobbiamo pensare a una base di dati relazionale come a un **archivio** in cui sono contenuti dei **dati strutturati sotto forma di tabelle**, ognuna delle quali è suddivisa a sua volta in diversi **"campi"**. La nostra "conoscenza" quindi sarà espressa sotto questa particolare forma: informazioni **"non strutturate"** diventano, attraverso un database, informazioni **"strutturate"** (non vi preoccupate, tra poco faremo alcuni esempi concreti!).

Abbiamo quindi più o meno compreso cos'è una base di dati. Veniamo ora al secondo termine, **"relazionale"**. Che cosa significa? Il tipo di database di cui ci occuperemo si fonda su un **modello** che prevede che le informazioni contenute nella nostra base di dati siano rappresentate come una serie di **relazioni tra diversi dati**, e in particolare tra **attributi e valori**. Ne consegue quindi che il modo per meglio rappresentare questa **relazione** è, come abbiamo detto poco fa, la **tabella**.

Il modello relazionale non è molto "recente", perché è stato formulato nel 1970 da **Edgar Codd**, un informatico inglese, e conosce diverse "implementazioni", ovvero diversi sistemi per metterlo in pratica. Tali implementazioni sono dette anche **"DataBase Management System"** (abbreviato in **DBMS**, "sistema di gestione di basi di dati"), e **MySQL** non è altro che un DBMS.

Facciamo un esempio che esula un po' (ma neanche più di tanto) dal contenuto del nostro ebook per fissare meglio i concetti di cui abbiamo parlato fin qui. Pensiamo a un **romanzo**: è un tipo di informazione **poco strutturata** (giusto un indice con qualche

capitolo numerato). Il vantaggio dell'informazione non strutturata è la sua **grande flessibilità**, ma il grande svantaggio è che è **difficile trarne informazioni**.

Se per esempio vogliamo conoscere la **frequenza** di un particolare **lemma** all'interno del romanzo, non ci riusciremo (a meno che non lo leggiamo da cima a fondo e ci annotiamo il numero di volte in cui il nome compare, ma è un'operazione **inutilmente laboriosa**). Per lemma si intende la **forma-base** alla quale viene riportato un insieme di forme (per esempio, per l'insieme di forme *vado-vai-andiamo-andarono* il lemma è **"andare"**).

Come facciamo quindi a sapere, per esempio, quante volte viene usato il verbo "andare" nei "Promessi Sposi"? Abbiamo bisogno di **strutturare l'informazione** contenuta nel nostro romanzo. Possiamo quindi rappresentare l'**intero romanzo** sotto forma di **base di dati relazionale**, e lo facciamo creando un semplice database con due tabelle, di questo tipo:

Tabella 1 - Capitoli

Id	Num	PersonaggioPrincipale	PersonaggiSecondari	Descrizione	TestoHTML
1	I	Don Abbondio	I bravi, Perpetua	Don Abbondio incontra i bravi, che gli intimano di non celebrare il matrimonio tra Renzo e Lucia.	Quel ramo del < href="#nota1">lag di Como, ch volge mezzogiorno…

Tabella 2 – Forme

Id	IdCapitolo	Forma	Lemma	Categoria	Informazioni
1	1	Quel	QUELLO	Aggettivo	Dimostrativo, maschile singolare
2	1	ramo	RAMO	Sostantivo	Comune, maschile singolare

3	1	del	DI	Preposizione	Articolata	
4	1	lago	LAGO	Sostantivo	Comune, singolare	maschile
5	1	di	DI	Preposizione	Semplice	
6	1	Como	COMO	Sostantivo	Proprio	

E così via. Abbiamo quindi un esempio di **informazione strutturata** attraverso un **database relazionale** fondato sull'uso di tabelle che rappresentano le informazioni tramite coppie attributo-valore. **Attributi** sono, per esempio, *forma, lemma, categoria* e valori sono *quel, ramo, QUELLO, aggettivo* e così via. Il grande vantaggio che una base di dati ci offre sta nel fatto che possiamo, come avrete intuito, **interrogarla per estrarre informazioni** tramite **appositi comandi** (di cui parleremo nei prossimi capitoli).

Facciamo un esempio che ci interessa più da vicino: vediamo un esempio di database relazionale per la **gestione di un blog**. Poniamo un **minimo di tre tabelle**: una per i post, una per i commenti e una per gli utenti.

Tabella 1 – Post

Id	IdAutore	Titolo	TestoHTML	Data	Link
1	1	Salve a tutti!	Salve a tutti! Questo è il mio primo post sul blog.	2011-04-01 12:00:00	PrimoProgetto/post1.php

Tabella 2 – Commenti

Id	IdAutore	IdPost	Titolo	TestoHTML	Data
1	2	1	Ciao!	Ciao! Molto bello il tuo nuovo blog!	2011-04-01 12:30:00
2	3	1	Saluti!	Che bello questo blog!	2011-04-02 11:00:00

Tabella 3 – Autori

Id	Nickname	Email	Nome	Sito
1	FedericoG	aaa@afdfaxs.xx	Federico	losnaweb.com
2	IlariaB	dsds@dsdsds.yy	Ilaria	finestresullarte.info

Questo è, in modo molto semplice e schematico, un modo per **rappresentare la struttura** del nostro **primo blog**. La flessibilità del modello relazionale ci dà l'opportunità di creare le basi di dati **più complesse** a seconda delle nostre esigenze: quelli che abbiamo visto sono soltanto due esempi che ci mostrano come qualsiasi tipo di informazione possa essere strutturata.

È possibile anche unire due o più tabelle (per esempio la tabella "post" e la tabella "autori), tramite appositi comandi, per **costruire ancora più relazioni** (per esempio, se vogliamo far comparire vicino al post anche il nickname del suo autore, nel nostro caso).

Chiudiamo con alcune informazioni su **MySQL** in particolare. Abbiamo specificato prima che si tratta di un DBMS, ovvero di un sistema di gestione di basi di dati, fondato sul modello relazionale. È un software **open source** (quindi è **del tutto gratuito**), è supportato sia da **Windows**, sia da **Linux**, sia da **Mac**, ed è attualmente il DBMS **più diffuso al mondo**. Alcuni si riferiscono a MySQL chiamandolo "linguaggio MySQL": **attenzione** a come si usano i termini, perché "linguaggio" non è il termine corretto. MySQL è un DBMS che sfrutta **la sintassi del linguaggio SQL**, che cominceremo a vedere, come accennato, dai prossimi capitoli.

SQL sta per **"Structured Query Language"**, che significa **"Linguaggio Strutturato di Interrogazioni"** (le **"query"** sono le **interrogazioni al database**). Ultima cosa, una curiosità: come si pronuncia MySQL? C'è chi lo pronuncia all'inglese, "mai-es-chiù-el", quelli che vogliono fare i "fighi" lo pronunciano come se fosse scritto "My Sequel" (la prima versione del linguaggio SQL infatti si chiamava **SEQUEL** e nacque negli anni Settanta… anche se spesso molti cannano la pronuncia della parola "sequel" eheh), c'è chi, come me, la pronuncia con un anglo-italico "mai-esse-cu-elle".

Come avevamo detto durante il secondo capitolo, il programma che utilizzeremo per amministrare il nostro database sarà **PhpMyAdmin**. Come funziona questo programma? Lo vedremo nel prossimo capitolo!

21. Costruiamo il database

Dopo una **panoramica teorica** su **MySQL**, è finalmente arrivato il momento di **costruire il nostro database**. Lo faremo direttamente tramite il programma **PhpMyAdmin**, senza utilizzare comandi. Per prima cosa avviamo il nostro **Wamp**, facciamo clic con il tasto sinistro sull'icona nella barra delle applicazioni e quindi, dal menù, selezioniamo la seconda voce: **PhpMyAdmin**.

Ci si aprirà la pagina del browser che avevamo scelto durante le fasi di installazione di Wamp. Se avete il programma in italiano, noterete che più o meno al centro della pagina c'è una sezione con scritto "**MySQL localhost**" e, subito sotto, la scritta **"Crea un nuovo database"** sotto la quale compaiono due campi. In quello vuoto, inseriamo il **nome del nostro database** (per esempio, "mioblog"). Il secondo è un **select**, che serve per selezionare la **"collation"**, ovvero **il set di caratteri** del nostro **database**: andrà bene un *utf8_general_ci*.

Possiamo quindi fare clic sul tasto *crea*: adesso abbiamo un **database**, chiamato "mioblog", ovviamente vuoto perché dobbiamo ancora **creare le tabelle**. Portiamoci quindi dentro la pagina del database: vediamo un menù con una serie di voci "struttura", "sql", "cerca", "query da esempio", "esporta", "importa", "operazioni", "privilegi", "elimina". Vedremo dopo a cosa servono queste voci: adesso è il momento di creare **una tabella**.

Lo facciamo tramite **il form** che vediamo **al centro della pagina**. Supponiamo quindi di voler creare la tabella "post" come abbiamo visto **nell'ultimo capitolo**. Inseriamo **nel primo campo il nome della tabella**, e **nel secondo il numero di campi** (cioè degli **attributi** a cui andremo ad associare i **valori**): nel nostro caso, i campi sono sei (uno per l'id, uno per l'id dell'autore, uno per il titolo, uno per il testo formattato in linguaggio HTML, uno per la data e uno per il link al post).

A questo punto ci comparirà un form con un numero di righe uguale al numero che abbiamo immesso poc'anzi, e per ogni riga notiamo una serie di colonne, di cui adesso comprendiamo il significato:

. **Campo**: è il nome del campo. Nel caso della nostra tabella, il primo sarà "Id", il secondo "IdAutore", il terzo "Titolo" e così via. Il **consiglio che vi do** è quello di utilizzare, per i campi, sempre **caratteri alfanumerici** o al più **underscore** se vogliamo inserire spazi. Sono **da evitare** caratteri accentati e caratteri speciali. In più, **non è possibile** assegnare ai nomi dei campi **le parole chiave che definiscono i comandi SQL** (sono un gruppo di parole come SELECT, DELETE, LIKE, LIMIT, JOIN e altre che conosceremo nei prossimi capitoli);

. **Tipo**: è il tipo del campo. I principali sono: **INT** (numero intero), **VARCHAR** (stringa di caratteri a lunghezza variabile), **DATE** e **DATETIME** (per le date: il secondo tipo può contenere anche l'ora), **TEXT**, **MEDIUMTEXT** e **LONGTEXT** (testi per un massimo di, rispettivamente, 2^16, 2^24 e 2^32 caratteri), **DECIMAL** (per i numeri decimali). Nel nostro caso inseriremo i tipi **INT** per Id, IdAutore e NumCommenti, **VARCHAR** per Titolo e Link, **DATETIME** per data e **TEXT** per TestoHTML ;

. **Lunghezza/Set**: la lunghezza massima dei campi. Per i campi numerici tipicamente si inserisce 11 come lunghezza (ovvero il numero massimo che un campo di tipo INT può avere come lunghezza: se lasciamo in bianco verrà apposto automaticamente proprio 11), mentre per i nostri campi VARCHAR possiamo scegliere una lunghezza massima di 100 caratteri. Lasciamo in bianco DATETIME e TEXT;

. **Predefinito, Collation, Attributi e Null**: possiamo tralasciare queste colonne per cui le lasciamo in bianco (per la cronaca, servono rispettivamente: se vogliamo dare un **valore predefinito al campo**, se vogliamo dare al campo un **set di caratteri diverso** rispetto a quello dell'**intero database**, se vogliamo specificare **attribuiti particolari** -per esempio se vogliamo assegnare al campo una **collation binaria**- e se vogliamo associare, tra gli altri, anche il valore **NULL** al nostro campo).

. **Indice**: serve per impostare l'indice al campo. A noi interessa l'indice **"PRIMARY"**, che serve per identificare il campo **"più importante"** della tabella, il cui valore deve essere univoco (non devono cioè esserci altri record con lo stesso valore per quel campo). Solitamente è il campo Id;

. **AUTO_INCREMENT**: questo è molto importante. Serve per **far aumentare automaticamente di una unità** il campo ogni volta che verrà inserito **un nuovo record**. Lo spuntiamo solo per il campo Id;

. **Commenti**: se vogliamo aggiungere commenti per un campo.

Una volta creata la nostra prima tabella **possiamo creare tutte le altre** e cominciare a svolgere su di esse **alcune operazioni**. Prima però vediamo cosa significano quelle voci di MySQL di cui abbiamo parlato prima:

. **Struttura**: ci mostra la **struttura del database**, ovvero l'elenco delle tabelle;

. **SQL**: contiene un campo che ci permette di eseguire **sintassi SQL** (che inizieremo a vedere dal prossimo capitolo);

. **Cerca**: per cercare una parola, un valore nel database;

. **Query da esempio**: se non conosciamo la sintassi SQL, questa voce ci aiuta a fare interrogazioni al database;

. **Esporta**: per esportare il database (possiamo scegliere tra tanti formati);

. **Importa**: se abbiamo un database (solitamente un file con estensione .sql)

possiamo importarlo;

. **Operazioni**: se vogliamo inserire una nuova tabella, rinominare il database, copiarlo o cambiare la collation;

. **Privilegi**: se ci sono più utenti che utilizzano PhpMyAdmin, possiamo impostare da qui i loro privilegi in termini di operazioni che possono eseguire sui database.

Vediamo ora quali sono **le operazioni** che si possono fare **sulle diverse tabelle**. Clicchiamo quindi su una delle nostre tabelle (ricordo che l'elenco si può trovare nella pagina **"Struttura"**): vedremo un'altra serie di voci che costituiscono un menù. Vediamo a cosa servono:

. **Mostra**: cliccando su questa voce potremo vedere **tutti i record del database**, e possiamo **modificarli velocemente** (facendo clic sull'**icona a forma di penna**, attraverso la quale ci si aprirà un form simile a quello della voce **"Inserisci"** che vedremo tra poco) o eliminarli (facendo clic **sull'icona a forma di croce**: prima ci verrà chiesta la conferma dell'operazione);

. **Struttura**: ci mostra la **struttura della tabella**. I campi possono essere **modificati o eliminati**;

. **SQL**: per eseguire **sintassi SQL** sulla tabella;

. **Cerca**: per cercare un **valore** nella tabella;

. **Inserisci**: per inserire **record** nella tabella. Ci apparirà un form in cui dovremo inserire un **valore per ogni attributo** e quindi fare clic su "esegui";

. **Esporta e Importa**: per esportare e importare la tabella;

. **Operazioni**: da qui possiamo riordinare, spostare, copiare, rinominare, commentare, analizzare la tabella;

. **Svuota**: per cancellare **tutti i record della tabella**;

. **Elimina**: per cancellare **tutta la tabella**.

Gestire i database con **PhpMyAdmin**, una volta appresi questi concetti, diventa, ovviamente con l'esercizio e con la pratica, **molto semplice e intuitivo**.

Come ultimo argomento vediamo come **connetterci al database** attraverso il nostro sito in **PHP**. Dobbiamo innanzitutto creare **quattro variabili**, che ci servono per **memorizzare l'indirizzo del server**, il **nome del database**, il **nome dell'utente** e la **password dell'utente del database**. Tipicamente, quando si lavora in locale, il server è **"localhost"** e il nome dell'utente è **"root"**:

```
$host = ‘localhost’;

$utente = ‘root’;

$password = ‘password’;

$db = ‘primoprogetto’;
```

A questo punto dobbiamo quindi invocare la funzione **mysql_connect**, che serve, come avrete intuito dal nome, a far sì che il nostro sito **possa connettersi al server MySQL**. La funzione riceve come parametri **l’indirizzo del server, il nome dell’utente e la sua password**:

```
$connessione = mysql_connect($host,$utente,$password);
```

Tipicamente si usa “estendere” la funzione **mysql_connect** con la funzione **die**, che serve per **fermare lo script in caso di errori** e per stampare di conseguenza **un messaggio all’utente**. Si invoca subito dopo *mysql_connect*, a cui viene unita tramite la parola **“or”**:

```
$connessione = mysql_connect($host,$utente,$password) or die(“Connessione non riuscita”);
```

Se non avverrà una connessione al server MySQL per un qualsiasi errore (per esempio, se abbiamo inserito la password sbagliata), ci sarà un **warning** e verrà stampata la scritta “Connessione non riuscita”. Abbiamo associato il risultato di questa funzione a una variabile perché è possibile chiudere la connessione tramite la funzione **mysql_close**, che riceve come parametro proprio il risultato di *mysql_connect*:

```
mysql_close($connessione);
```

È bene specificare che la connessione **si chiude comunque quando si arresta lo script** che la esegue: tuttavia *mysql_close* può servirci se vogliamo chiudere noi la connessione **in un qualche punto preciso** del nostro sito.

A questo punto, dopo esserci **connessi al server**, dobbiamo selezionare il database che ci interessa, e lo facciamo tramite la funzione **mysql_select_db**, che riceve come parametro **il nome del nostro database** (che avevamo memorizzato nella variabile $db):

```
mysql_select_db($db);
```

Bene! Adesso siamo collegati al nostro database. Le documentazioni ufficiali delle funzioni che abbiamo visto nel capitolo si trovano qui: http://php.net/manual/en/function.mysql-connect.php (mysql_connect), http://it.php.net/manual/en/function.die.php (die), http://it.php.net/manual/en/function.mysql-close.php (mysql_close), http://it.php.net/manual/en/function.mysql-select-db.php (mysql_select_db).

Quindi, ricapitolando, il codice per connetterci al database è questo:

```
<?php



$host = 'localhost';

$utente = 'root';

$password = 'password';

$db = 'primoprogetto';

$conn = mysql_connect($host,$utente,$password) or die("Connessione non riuscita");

mysql_select_db($db);



?>
```

E dovremo utilizzarlo **in ogni singola pagina del sito** per la quale è prevista una interazione con il database, quindi **il consiglio** che vi do è quello di **creare un file esterno** in cui inserirete questi dati, e poi lo richiamerete in ogni pagina semplicemente con un *include*. C

22. Operazioni sul database: INSERT, UPDATE e DELETE

Iniziamo una **“gita”** di tre capitoli all’interno delle **operazioni che si possono fare sul database**, cercando di fornire una panoramica abbastanza completa per poter gestire un sito **PHP/MySQL** di una certa complessità. In questo capitolo vedremo **tre operazioni sul database**, e in particolare vedremo come **inserire**, **aggiornare** e **cancellare** record.

Partiamo dalla prima: l’operazione **INSERT**. No, non è una parola urlata: i comandi **SQL** si scrivono **tutti maiuscoli** (in **PHP** si possono comunque scrivere **tutti minuscoli** e non succede niente di strano: è sempre però bene scriverli tutti maiuscoli in primo luogo perché così prevede la **documentazione ufficiale**, in secondo luogo per **non rischiare di fare confusione** e in terzo luogo perché è sempre bello **mantenere una forma elegante)**!

Prima però dobbiamo fare una **piccola introduzione**. Nello scorso capitolo abbiamo detto che le **interrogazioni al database** si chiamano **“query”** (anche le operazioni di **inserimento**, **aggiornamento** e **cancellazione** si chiamano “query”). Per gestire le *query* inviate al database esiste una specifica funzione PHP, che si chiama **mysql_query** e che riceve come parametro proprio la *query*. Ma cos’è in sostanza una *query* parlando solo di PHP? Non è altro che **una stringa di testo**, ma una stringa **un po’ particolare**, perché contiene nient’altro che le **istruzioni SQL**.

Vediamo quindi come funziona il tutto. L’operazione di inserimento prevede che venga scritto prima il comando **INSERT INTO**, quindi il nome della tabella all’interno della quale inserire i dati, poi la **clausola VALUES** seguita, tra parentesi tonde, dai **dati da inserire** scritti **nell’ordine di come appaiono i campi nella tabella**. Prendiamo come punto di riferimento la tabella *Post* che abbiamo creato nel capitolo precedente: dovremo seguire, in fase di inserimento, l’ordine Id – IdAutore – Titolo – TestoHTML – Data – Link. Questo è ciò che dovremo quindi scrivere:

```
$query = “INSERT INTO Post VALUES (‘NULL’, ‘1’, ‘Ciao a tutti’, ‘Ciao a tutti! Questo è il mio primo post’, ‘2011-04-03 12:00:00’, ‘/post/primopost’)”;
```

Se invertissimo l’ordine (per esempio scambiando “1” con “Ciao a tutti”) il risultato sarebbe l’inserimento di “Ciao a tutti” nel campo IdAutore e “1” nel campo Titolo: **ci sarebbe quindi un errore**, perché abbiamo assegnato a IdAutore il tipo INT, e “Ciao a tutti” non è davvero **un numero intero**. Quindi bisogna fare **grande attenzione** a **rispettare l’ordine dei campi** in questa operazione (**sbagliarsi non è difficile**). Un’altra accortezza: le stringhe di testo vanno sempre messe **tra apici o doppi apici**. Per quanto riguarda i campi di tipo **INT**, si possono anche lasciare senza apici. Mi raccomando, perché se non inserite le stringhe tra apici, **la query non verrà eseguita**. Infine, l’ultima

cosa di cui tener conto. Abbiamo inserito il valore **NULL** nel campo Id perché si tratta di un campo a cui abbiamo assegnato **l'auto-incremento**: non sarà quindi necessario specificare il numero, perché verrà a**utomaticamente inserito** il **primo numero libero disponibile**. Per esempio, se questo è il primissimo post del nostro blog, nel campo Id sarà inserito il numero zero.

Bene, detto questo e una volta dichiarata la variabile $query, che non è altro che una stringa di testo, come abbiamo detto, dobbiamo invocare la funzione *mysql_query*:

```
mysql_query($query);
```

Fatte queste operazioni, i dati **verranno inseriti nel database**. Mi preme specificare che la funzione *mysql_query* è da utilizzarsi **per qualsiasi operazione eseguita sul database**, quindi sarà una funzione che ci accompagnerà per un bel po' di tempo. Potete trovare la documentazione ufficiale a questa pagina: http://php.net/manual/en/function.mysql-query.php.

Abbiamo visto come inserire dati che **riempiano tutti i campi della tabella**. Ma cosa dovremmo fare se volessimo riempire soltanto alcuni campi? Per esempio se volessimo lasciare in bianco il campo Link? Dovremmo specificare **i singoli campi subito dopo il nome della tabella**, inserendoli tra parentesi tonde, come nell'esempio seguente:

```
$query = "INSERT INTO Post (Id, IdAutore, Titolo, TestoHTML, Data) VALUES ('NULL', '1', 'Ciao a tutti', 'Ciao a tutti! Questo è il mio primo post', '2011-04-03 12:00:00')";

mysql_query($query);
```

In questo modo verranno inseriti tutti i dati che abbiamo specificato ma sarà **lasciato in bianco** il campo Link. Questa sintassi è più o meno simile per **tutte le operazioni SQL**: abbiamo un **comando** e abbiamo poi una o più **clausole** che specificano meglio ciò che viene indicato dal comando.

Passiamo ora al comando **UPDATE**, che serve per **aggiornare i record** e che è un pochino più complesso rispetto a INSERT. Partiamo dal **caso base**, quello per aggiornare un solo campo, ma in tutti i record:

```
$query = "UPDATE Post SET IdAutore = 1";
```

```
mysql_query($query);
```

Questo è il caso più banale (e, se vogliamo, anche il più inutile): con il comando UPDATE abbiamo **aggiornato** la tabella Post impostando **tutti i valori** del campo IdAutore (quindi su tutti i record) a 1. Per aggiornare **soltanto un campo nella tabella** (o due o tre o comunque **non tutti**), dovremo essere **più specifici**: ci servirà la clausola **WHERE**, tramite la quale specificheremo **un altro campo** che dovrà identificare bene **il record (o i record) da aggiornare**. Ma vedendo il codice è più facile da intuire:

```
$query = “UPDATE Post SET Titolo = ‘Secondo post’ WHERE Id = ‘1’”;

mysql_query($query);
```

In questo modo **abbiamo modificato un solo record**, quello che ha come Id il numero uno (ricordiamo che Id è la **chiave primaria** della nostra tabella quindi si tratta di un campo che **non può essere doppio**, non ci saranno cioè due record **con lo stesso valore nel campo Id**). Una piccola parentesi: noi stiamo facendo le operazioni **con due passaggi** ma nessuno vi vieta di fare le interrogazioni in questo modo:

```
mysql_query(“UPDATE Post SET Titolo = ‘Secondo post’ WHERE Id = ‘1’”);
```

È un modo più rapido per fare **la stessa cosa**: in questo corso facciamo però uso delle variabili per **una più chiara lettura e comprensione della sintassi SQL**.

Ma andiamo avanti con le nostre operazioni. Se nel nostro database avessimo due post che hanno come titolo “Ciao” e volessimo cambiarlo in “Ciao a tutti!” dovremmo comportarci come sopra:

```
$query = “UPDATE Post SET Titolo = ‘Ciao a tutti!’ WHERE Titolo = ‘Ciao’”;

mysql_query($query);
```

WHERE è una delle **clausole più importanti** della sintassi SQL perché si abbina **a moltissimi comandi**. Cosa dobbiamo fare invece se vogliamo dare **lo stesso titolo** a **due post che hanno titoli diversi**, per esempio uno “Ciao” e uno “Ciao a tutti!”? Introduciamo un’altra clausola, **OR**:

```
$query = “UPDATE Post SET Titolo = ‘Ciao a tutti quanti!!!’ WHERE Titolo = ‘Ciao’ OR Titolo = ‘Ciao a tutti!’”;

mysql_query($query);
```

La clausola OR serve per **“legare”** tra di loro due espressioni introdotte dalla clausola WHERE, e possiamo usarla, all’interno della stessa query, **tutte le volte che vogliamo** (anche tre o quattro o più volte:

```
$query = “UPDATE Post SET Titolo = ‘Ciao a tutti quanti!!!’ WHERE Titolo = ‘Ciao’ OR Titolo = ‘Ciao a tutti!’ OR Titolo = ‘Altro post’ “;

mysql_query($query);
```

Cosa fare invece quando abbiamo tre post con titolo “Ciao a tutti!”, scritti **da due autori diversi**, e noi vogliamo cambiare il titolo **solo a quelli scritti da un certo autore**? Dobbiamo utilizzare la clausola **AND**:

```
$query = “UPDATE Post SET Titolo = ‘Ciao a tutti quanti!!!’ WHERE Titolo = ‘Ciao a tutti!’ AND IdAutore = 1”;

mysql_query($query);
```

In questo modo abbiamo impostato il titolo a tutti quei post che hanno come titolo “Ciao a tutti!” **e che sono stati scritti** dall’autore contrassegnato dall’id 1. Quindi se ci sono altri post che hanno per titolo “Ciao a tutti!” (scritti ovviamente da autori diversi) o se ci sono altri post scritti dallo stesso autore (con titolo diverso), bene, tutti questi post **non saranno aggiornati**: l’operazione sarà valida solo per i post che **soddisfano le clausole WHERE e AND**.

Ovviamente poi, nelle espressioni delle clausole, è possibile usare, oltre all’uguale, anche **altri operatori** (maggiore, minore, maggiore-uguale, minore-uguale). Per esempio, volendo cambiare il titolo a tutti i post che hanno un Id maggiore di 5:

```
$query = “UPDATE Post SET Titolo = ‘Ciao’ WHERE Id > 5”;

mysql_query($query);
```

Introduciamo poi un’**ulteriore clausola**, che ci serve se vogliamo prendere un certo numero di **record contigui**. Per esempio, prendiamo il caso che abbiamo visto adesso:

vogliamo cambiare il titolo a **tutti i post** che hanno un id **maggiore di 5**, ma arrivando solo **fino a quello che ha come id 8** (quindi valido solo per i primi tre record). Vogliamo quindi **non aggiornare** i post che hanno id 9, 10, 11 eccetera. Ci serve la clausola **LIMIT**, che deve essere seguita **da un numero** che **indica il numero di post da tenere in considerazione**:

```
$query = “UPDATE Post SET Titolo = ‘Ciao a tutti!!!’ WHERE Id > 5 LIMIT 3”;

mysql_query($query);
```

In questo modo aggiorniamo sì i post che hanno un id maggiore di 5, ma solo i tre immediatamente successivi grazie alla clausola LIMIT. Verranno aggiornati quindi solo i post 6, 7 e 8.

Vediamo infine l’ultima istruzione, **DELETE**, che serve per **cancellare record dal nostro database**. Per cancellare un singolo record dal database la sintassi è molto semplice:

```
$query = “DELETE FROM Post WHERE Id = ‘1’”;

mysql_query($query);
```

Abbiamo semplicemente cancellato il post che ha come id il numero 1. Possiamo **applicare anche a DELETE** tutte le **clausole** che abbiamo visto per **UPDATE**. Vediamo per esempio con le clausole OR e AND:

```
$query = “DELETE FROM post WHERE Id = 5 OR Id = 6”;

mysql_query($query);

$query2 = “DELETE FROM post WHERE Titolo = ‘Ciao a tutti’ AND IdAutore = 3”;

mysql_query($query2);
```

Nel primo caso abbiamo cancellato i post contrassegnati dagli id 5 e 6, nel secondo caso abbiamo invece cancellato tutti i post che hanno per titolo “Ciao a tutti” e che sono stati scritti dall’autore contrassegnato dall’id numero 3.

È possibile anche **cancellare tutta la tabella**:

```
$query = “DELETE FROM post”;

mysql_query($query);
```

Attenzione però! Una volta eseguito lo script **non è più possibile arrestarlo o tornare indietro**, quindi mi raccomando di usarlo con parsimonia, **solo se si è sicuri del risultato** e dove **non fa danni**.

Bene, siamo arrivati alla fine e vi lascio con un esercizio. Costruite una vera pagina per scrivere post in un blog, nel senso che abbiamo un form con una serie di opzioni, e una volta compilato il tutto, i dati immessi dal nostro utente vengono inviati al nostro database. Per la stampa della data potete utilizzare la funzione *date* che abbiamo visto nel capitolo in cui abbiamo costruito il guestbook.

23. Operazioni sul database: SELECT e JOIN

Continuiamo la panoramica sulle **operazioni** che si possono fare sul **database** vedendo in questo capitolo **due delle operazioni più importanti**: la prima serve per **estrarre record** (e quindi **informazioni**) dalle tabelle, la seconda invece serve per **unire due o più tabelle insieme**.

La prima di queste due operazioni si compie utilizzando il comando **SELECT**. Proviamo per esempio a selezionare il campo Titolo e il campo Link dalla tabella post:

```
$query = “SELECT Titolo, Link FROM Post”;
```

La sintassi è quindi semplice: basta scrivere **SELECT** seguito dal **nome dei campi** da cui vogliamo **estrarre informazioni**, ognuno **separato dall’altro** attraverso **una virgola**, e quindi, dopo la parola **FROM** scrivere il **nome della tabella**. E se volessimo invece selezionare **tutti i campi**? Invece di scrivere i nomi di tutti quanti i campi, basta semplicemente inserire **un asterisco tra SELECT e FROM**:

```
$query = “SELECT * FROM Post”;
```

E ovviamente, anche con SELECT, possiamo utilizzare **le clausole WHERE, AND e OR**:

```
$query = “SELECT * FROM Post WHERE Titolo = ‘Ciao’”;

$query2 = “SELECT * FROM Post WHERE Titolo = ‘Ciao’ AND IdAutore = 1”;

$query3 = “SELECT * FROM Post WHERE Titolo = ‘Ciao a tutti’ OR Titolo = ‘Ciao’”;
```

Nel primo caso abbiamo selezionato tutti i campi della tabella Post dai record il cui titolo è “Ciao”, nella seconda query stessa cosa ma i post devono essere stati scritti dall’autore contrassegnato dall’id numero 1, e nella terza interrogazione abbiamo invece selezionato tutti i campi dei record che hanno come titolo “Ciao a tutti” oppure “Ciao”.

Possiamo poi anche **dare un ordine ai record** che estraiamo con il comando SELECT, utilizzando la clausola **ORDER BY** seguita dal campo in base al quale vogliamo ordinare le informazioni. Se per esempio vogliamo ordinarli in base alla data:

```
$query = “SELECT * FROM Post WHERE Titolo = ‘Ciao’ AND IdAutore = 1 ORDER BY Data
DESC”;
```

E possiamo scegliere se **ordinare in modo decrescente**, come nel caso dell’esempio (**DESC**) oppure per **ordinare in modo crescente** (**ASC**).

In genere il SELECT si usa, come abbiamo detto all’inizio, per **estrarre informazioni**: e tali informazioni andranno poi **mostrate all’utente**. Bene, come si fa per mostrare queste informazioni all’utente? Abbiamo bisogno di una nuova funzione, **mysql_fetch_array**. Riceve come **primo parametro** il **risultato della funzione mysql_query**, e come **secondo parametro il tipo di array** che vogliamo utilizzare, perché il risultato della nostra *query* **sarà trasformato in un array**. Possiamo quindi scegliere un **array numerico (MYSQL_NUM)**, un **array associativo (MYSQL_ASSOC)** o **entrambi (MYSQL_BOTH)**. Il mio consiglio è quello di utilizzare un **array associativo**: gli **indici** saranno nient’altro che i **nomi dei campi**. Se **non volete specificare** il tipo di array, sarà assegnato quello di default, ovvero **MYSQL_BOTH**.

E dal momento che avremo bisogno di creare **un array per ogni record del database**, dovremo utilizzare **una iterazione di tipo while**, e per stampare ciò che ci interessa dovremo **tener conto** del fatto che staremo **operando su di un array**:

```
$query = “SELECT * FROM Post WHERE IdAutore = 1”;

$risultato = mysql_query($query);

while ($record = mysql_fetch_array($risultato, MYSQL_ASSOC))

{

    echo “{$record[‘Titolo’]}<br />”;

}
```

Attraverso **il nostro while**, nel caso dell’esempio, con una stampa scriviamo nella pagina il titolo dei post che soddisfano le condizioni poste dal comando SELECT.

Bene, facciamo ora un altro esempio. Supponiamo di voler scrivere, da qualche parte nella home page del nostro blog, **i titoli dei post accompagnati dal nickname dell’autore**. Nella tabella post il nickname dell’autore degli articoli non è presente: abbiamo soltanto

l’id. Come facciamo quindi a compiere questa operazione? Dovremo necessariamente **unire la tabella Post con la tabella Autori**, per fare in modo che ai campi della tabella Post vengano anche aggiunti i campi della tabella Autori.

Per fare questo utilizziamo il comando **JOIN**, ma ci deve essere **una condizione**, ovvero **la base su cui uniremo le tabelle**. Cosa significa? Significa che dovrà esserci **un campo che avrà gli stessi valori per entrambe le tabelle**. Nel nostro caso è il campo dell’id dell’autore, ovvero IdAutore nella tabella Post e Id nella tabella Autori: affinché le due tabelle possano essere unite, ci deve essere **una corrispondenza tra IdAutore di Post e Id di Autori**. Questo significa che a tutti i record che hanno un certo IdAutore **verranno abbinati i valori** di quei record della tabella Autori il cui Id è lo stesso di IdAutore.

Nel nostro database l’autore 1 è Federico. Con il comando JOIN, a tutti i post scritti da Federico memorizzati nella tabella Post e contrassegnati, ovviamente, con l’IdAutore uguale a 1, saranno affiancati i record della tabella Autori il cui Id è uguale a 1.

Vediamo quindi la sintassi:

```
$query = “SELECT * FROM Post JOIN Autori ON Post.IdAutore = Autori.Id”;

$risultato = mysql_query($query);
```

Quindi subito dopo il nome della tabella inseriamo **il comando JOIN** seguito **dal nome della tabella da unire**. Inseriamo poi **la clausola ON** seguita dai **due campi delle tabelle che devono corrispondere tra di loro**: nel nostro caso IdAutore per quanto riguarda Post e Id per quanto riguarda Autori. Deve essere scritto prima il nome della tabella, quindi, **dopo un punto**, il nome del campo (*tabella.campo*).

Per stampare i risultati il metodo è ancora quello di prima:

```
while ($record = mysql_fetch_array($risultato, MYSQL_ASSOC))

{

   echo “{$record[‘Titolo’]}, scritto da {$record[‘Nickname’]}<br />”;

}
```

Supponiamo ora di voler stampare sia davanti al post che davanti all’autore il rispettivo id.

Per esempio in questa forma: “Post numero 2: Ciao a tutti!!!, scritto da FedericoG (autore numero 1)”. Se scrivessimo una cosa del genere:

```
echo “Post numero {$record[‘Id’]}: {$record[‘Titolo’]}, scritto da {$record[‘Nickname’]} (autore
numero {$record[‘Id’]})<br />”;
```

Ovviamente non otterremmo il risultato sperato, perché **ci sono due campi che hanno lo stesso nome** (Id), e quando succede questo viene preso il campo Id **dell’ultima tabella presa in considerazione** (nel nostro caso Autori) e scartato il campo **con lo stesso nome nelle tabelle precedenti**. Per ovviare a questo problema **possiamo rinominare**, all’interno della query, **il campo che crea il conflitto**. Lo facciamo tramite la **clausola AS**, che deve essere seguita **dal nuovo nome** del campo che ci interessa:

```
$query = “SELECT Post.Titolo, Post.Id, Autori.Id AS NumeroAutore, Autori.Nickname FROM Post
JOIN Autori ON Post.IdAutore = Autori.Id”;

$risultato = mysql_query($query);
```

Abbiamo quindi **rinominato il nome del campo** Id della tabella Autori in NumeroAutore (**ovviamente solo momentaneamente**, per la query: **non abbiamo alterato** in alcun modo la **struttura della tabella** all’interno del nostro **database**). Stampiamo poi in questo modo:

```
while ($record = mysql_fetch_array($risultato, MYSQL_ASSOC))

{

   echo “Post numero {$record[‘Id’]}: {$record[‘Titolo’]}, scritto da {$record[‘Nickname’]} (autore
numero {$record[‘NumeroAutore’]})<br />”;

}
```

Si possono poi **rinominare anche le tabelle**, magari per questioni di praticità e rapidità, e il sistema è lo stesso che abbiamo appena visto:

```
$query = “SELECT * FROM Post AS p JOIN Autori AS a ON p.IdAutore = a.Id”;

$risultato = mysql_query($query);
```

Le unioni che abbiamo visto fino a questo momento sono unioni di tipo **INNER JOIN**: significa che vengono **combinati tutti i record che hanno una corrispondenza** e vengono invece **scartati i record che non hanno corrispondenza**. Avremmo potuto specificare meglio le query viste finora (prendiamo per esempio l'ultima) scrivendo così:

```
$query = "SELECT * FROM Post AS p INNER JOIN Autori AS a ON p.IdAutore = a.Id";

$risultato = mysql_query($query);
```

Ma dal momento che si tratta del tipo di unione predefinito possiamo anche non specificarlo. Questo perché esiste anche un altro tipo di unione, **OUTER JOIN**, che al contrario delle unioni di tipo INNER JOIN tiene conto **anche dei risultati che non trovano corrispondenza tra di loro**. Le unioni OUTER JOIN si suddividono a loro volta in **LEFT JOIN**, ovvero unioni che contengono sempre **tutti i record della prima tabella** (la tabella di sinistra, *left*) mentre per quanto riguarda i record della seconda tabella (quella di destra, right) vengono estratti **solo i record che trovano una corrispondenza** con quelli della tabella di sinistra, e in **RIGHT JOIN**, che funzionano **all'inverso** rispetto alle unioni LEFT JOIN.

Per esempio, supponiamo di avere tre autori e quattro post: tre scritti dall'autore 1, uno scritto dall'autore 2 e nessuno scritto dall'autore 3. Prendiamo questa query:

```
$query = "SELECT * FROM Autori LEFT JOIN Post ON Autori.Id = Post.IdAutore";

$risultato = mysql_query($query);
```

In questo caso vengono selezionati tutti i record della tabella Autori uniti ai soli record della tabella Post che trovano una corrispondenza. Con un INNER JOIN l'autore numero 3 non sarebbe stato preso in considerazione dalla query, invece con questo OUTER JOIN di tipo LEFT (avremmo potuto scrivere, nel comando, anche **LEFT OUTER JOIN**) l'autore numero 3 viene **comunque estratto**. Se poi stampassimo:

```
while ($record = mysql_fetch_array($risultato, MYSQL_ASSOC))

{

   echo "{$record['Titolo']}, scritto da {$record['Nickname']}<br />";

}
```

Nel ciclo corrispondente all'autore numero 3 verrebbe stampato **uno spazio bianco**, perché **non esisterebbe** un elemento dell'array con indice Titolo abbinato all'elemento "Nickname" relativo all'autore 3.

Il RIGHT JOIN funziona allo stesso modo del LEFT JOIN ma **in maniera inversa**:

```
$query = "SELECT * FROM Post RIGHT JOIN Autori ON Autori.Id = Post.IdAutore";

$risultato = mysql_query($query);
```

In questo caso vengono estratti tutti i record della tabella di destra (Autori) mentre per quanto riguarda la tabella di sinistra (Post) vengono estratti solo i record che hanno una corrispondenza con Autori.

Vediamo **un'altra informazione importante**. **Specifichiamo sempre la clausola ON** altrimenti, se scrivessimo una cosa del genere:

```
$query = "SELECT * FROM Post JOIN Autori";

$risultato = mysql_query($query);
```

Otterremmo un **CROSS JOIN**, ovvero il **prodotto cartesiano delle due tabelle**, cosa che sicuramente, almeno ai fini del nostro blog, non ci interessa.

Possiamo poi unire anche **tre o più tabelle tra loro**. Per esempio, se avessimo un'ipotetica tabella DatiAutori in cui inserire indirizzi, numeri di telefono, immagini e quant'altro, potremmo scrivere una cosa del genere:

```
$query = "SELECT Post.Titolo, Autori.Nickname, DatiAutori.Email FROM Post JOIN Autori ON
Post.IdAutore = Autori.Id JOIN DatiAutori ON Autori.Id = DatiAutori.IdAutore";

$risultato = mysql_query($query);
```

E poi stampare in questo modo:

```
while ($record = mysql_fetch_array($risultato, MYSQL_ASSOC))
```

```
{

   echo “{$record[‘Titolo’]}, scritto da {$record[‘Nickname’]} (email: {$record[‘Email’]})<br /><br />”;

}
```

Abbiamo quindi ottenuto **un’unione di tre tabelle**, ma **non c’è limite** al numero di tabelle che possiamo unire.

Bene, siamo arrivati alla fine del capitolo e vi propongo con un esercizio. Create uno script per stampare, nella home page, il titolo dei post, il nome dell’autore e i titoli dei commenti relativi al post, in questa forma:

Post: Ciao a tutti!

Scritto da: Federico

Commenti…

Ciao!

Benvenuto!

Bellissimo sito!

24. Operazioni sul database: funzioni aggregate e query annidate

Terminiamo il nostro panorama sulle **operazioni** che si possono fare sul **database** parlando di **funzioni aggregate** e di **query annidate**. Iniziamo dalle prime, dette anche **"funzioni di aggregazione"**: si tratta di una serie di **funzioni** che il linguaggio **SQL** ci mette a disposizione per estrarre **diversi tipi di informazione** dal nostro database, perlopiù di tipo **statistico**.

Supponiamo, per esempio, di voler creare una pagina all'interno della quale inserire una serie di statistiche relative al nostro blog. Potremmo iniziare dal **numero complessivo** dei post presenti nel blog (e quindi nel database). Come fare per sapere quanti sono? Utilizziamo la funzione aggregata **COUNT**, che invocheremo sulla tabella dando un **nome al risultato** che verrà estratto tramite la clausola **AS**. Vediamo la sintassi:

```
$query = "SELECT COUNT(*) AS TotalePost FROM Post";

$risultato = mysql_query($query);
```

Tutte le funzioni devono essere invocate dopo il comando **SELECT**. Nel nostro caso, invochiamo la funzione COUNT **su tutti i campi** (*) e quindi chiamiamo il risultato TotalePost. Possiamo poi stampare il tutto come abbiamo visto finora:

```
while ($record = mysql_fetch_array($risultato, MYSQL_ASSOC))

{

   echo "Totale post presenti nel blog: "; echo $record['TotalePost'];

}
```

Supponiamo invece di voler conoscere il **numero totale dei commenti** ma **senza operare** sulla tabella *Commenti*: vogliamo utilizzare un campo *NumCommenti* della tabella *Post* in cui avremo inserito il numero di commenti per ogni singolo post. In questo caso ci sarà di aiuto la funzione **SUM**, che, come si può intuire dal nome, esegue la **somma dei valori dei record**. Supponiamo di avere quattro post: il primo ha dodici commenti, il secondo cinque, il terzo nove e il quarto quindici per un totale di quarantuno commenti.

La sintassi è del tutto simile a quella di COUNT. Invochiamo SUM subito dopo il comando SELECT, tra parentesi indichiamo **il campo che ci interessa** (ovvero il campo i cui valori saranno sommati tra di loro), diamo un nome al risultato e selezioniamo la

tabella:

```
$query2 = “SELECT SUM(NumCommenti) AS TotaleCommenti FROM Post”;

$risultato2 = mysql_query($query2);

while ($record2 = mysql_fetch_array($risultato2, MYSQL_ASSOC))

{

   echo “Totale commenti presenti nel blog: “; echo $record2[‘TotaleCommenti’];

}
```

A questo punto potrebbe essere interessante sapere qual è **il numero medio di commenti** ai post del nostro blog (nel nostro caso è 10,25). Abbiamo anche una funzione per conoscere la media: si tratta di **AVG** (dall’inglese “average” che significa “media”) e il meccanismo è identico a quello di SUM. Vediamo:

```
$query3 = “SELECT AVG(NumCommenti) AS MediaCommenti FROM Post”;

$risultato3 = mysql_query($query3);

while ($record3 = mysql_fetch_array($risultato3, MYSQL_ASSOC)) {

   echo “Media commenti ai post: “; echo $record3[‘MediaCommenti’];

}
```

Non è cambiato niente, se non la funzione invocata subito dopo il comando SELECT. Potremmo poi voler conoscere qual è il numero dei commenti del **post con il maggior numero di commenti** e al contrario qual è il numero di commenti del **post con il minor numero di commenti**. Per compiere queste due operazioni abbiamo le funzioni **MAX** e **MIN**, e anche in questi due casi il meccanismo è simile a quello delle funzioni che abbiamo appena visto. Diamo un’occhiata a MAX:

```
$query4 = “SELECT MAX(NumCommenti) AS Massimo FROM Post”;

$risultato4 = mysql_query($query4);

while ($record4 = mysql_fetch_array($risultato4, MYSQL_ASSOC))

{
```

```
  echo “Numero commenti massimo: “; echo $record4[‘Massimo’];

}
```

E adesso MIN:

```
$query5 = “SELECT MIN(NumCommenti) AS Minimo FROM Post”;

$risultato5 = mysql_query($query5);

while ($record5 = mysql_fetch_array($risultato5, MYSQL_ASSOC))

{

  echo “Numero commenti minimo: “; echo $record5[‘Minimo’];

}
```

Ma cosa dovremmo fare se volessimo sapere **qual è il post con il maggior numero di commenti** (e quindi non solo qual è il numero di commenti del post con il maggior numero di commenti)? Dobbiamo ricorrere a una **query annidata**, detta anche **“subquery”**: le query annidate sono **interrogazioni che si trovano all’interno di altre interrogazioni**. Il caso che prendiamo in esame noi è **il più semplice**: una interrogazione viene posta come **condizione della clausola WHERE**. Vediamo come funziona:

```
$query6 = “SELECT Titolo FROM Post WHERE NumCommenti = (SELECT MAX(NumCommenti)
FROM Post)”;

$risultato6 = mysql_query($query6);
```

Fino al WHERE quindi **niente di strano**: dopo l’uguale però inseriamo **un nuovo comando SELECT**, che dovrà estrarre **il numero massimo di commenti presenti in un post**. La query “più grande” estrarrà quindi il titolo il cui numero di commenti sarà uguale al massimo numero di commenti per un post presenti nel nostro database. Stampiamo poi molto semplicemente così:

```
while ($record6 = mysql_fetch_array($risultato6, MYSQL_ASSOC))

{

  echo “Post con il maggior numero di commenti: “; echo $record6[‘Titolo’];

}
```

Ripetiamo la stessa cosa anche per conoscere il post con il minor numero di commenti. Possiamo poi creare dei comandi SELECT che **estraggono informazioni dai risultati di altri comandi SELECT**, come in questo caso:

```
$query = “SELECT * FROM (SELECT * FROM Post WHERE Id > 5) AS PrimaSelect WHERE
NumCommenti > 6”;

$risultato = mysql_query($query);
```

Che è una complicazione inutile in quanto avremmo ottenuto lo stesso risultato con un solo comando SELECT, ma è utile per capire il meccanismo delle query annidate. In questo caso selezioniamo tutti i risultati tratti a loro volta dai risultati di una prima selezione, che dovrà obbligatoriamente essere nominata (AS PrimaSelect), altrimenti la query non funzionerà e ci sarà un errore. La prima selezione, quella interna, estrae tutti i record dalla tabella *Post* che hanno un id maggiore di 5, e da questa selezione, il comando SELECT principale estrae i post che hanno più di sei commenti.

A conclusione del capitolo, come sempre, un esercizio: create una query che esegua la somma di tutti i commenti presenti nel database escludendo però il post con il maggior numero di commenti.

25. Costruiamo un form di ricerca con PHP/MySQL

Con questo capitolo metteremo in pratica i concetti appresi negli ultimi capitoli dell'ebook e vedremo come costruire un semplice **form di ricerca PHP/MySQL** per fare ricerche all'interno del nostro blog. Come sempre, in occasione di capitoli a carattere "pratico", faremo un ripasso generale degli ultimi concetti ma introdurremo anche qualcosa di nuovo.

Allora, iniziamo a costruire il nostro form, in maniera molto semplice, come siamo abituati:

```
<form name="ricerca" action="ricerca.php" method="post">

<input type="text" name="cerca" />

<input type="submit" name="vai" value="Cerca!" />

</form>
```

Si tratta, come vedete, di un semplicissimo **form con metodo post** che rimanda a una pagina "ricerca.php". Il nostro form ha un campo, che servirà all'utente per **immettere il testo da ricercare**, e un **bottone per inviare i dati**.

Nel costruire la pagina "ricerca.php", come prima cosa assicuriamoci di **fare un controllo sul pulsante**: se è stato premuto, quindi se l'utente **proviene dalla pagina del form**, **sarà eseguito il codice**, e in caso contrario (se per esempio l'utente arriva alla pagina ricerca.php semplicemente digitandola nella barra degli indirizzi) **non dovrà succedere alcunché**. Vediamo:

```
if(isset($_POST['vai']))

{

   // codice

   // codice

   // codice

}

else
```

```
{

}
```

Vediamo ora come **riempire il blocco if**. Per prima cosa dobbiamo decidere se la ricerca riguarderà **soltanto i post o i commenti**. Noi vedremo tre esempi: il primo **per i post**, il secondo **per i commenti** e il terzo **per entrambi**.

Iniziamo dalla ricerca nei soli post. Per prima cosa stampiamo una scritta per indicare all'utente che **la ricerca è stata compiuta** e quindi inizializziamo una variabile, a cui **assoceremo la parola cercata**:

```
echo "Ecco i risultati della tua ricerca:<br /><br /> ";

$parolacercata = $_POST['cerca'];
```

Fatto questo è arrivato il momento di creare la nostra query per la ricerca all'interno della base di dati:

```
$query = "SELECT * FROM Post WHERE Titolo LIKE '%$parolacercata%' OR TestoHTML LIKE
'%$parolacercata%'";
```

Bene, vedete che abbiamo **due "novità"**: la prima è **l'operatore LIKE**, e la seconda sono **i due simboli di percentuale** prima e dopo la variabile. Procediamo con ordine.

LIKE è **un operatore di confronto**, come l'uguale, il maggiore, il minore e tutti gli altri. Funziona **più o meno come l'operatore di uguaglianza**, ma con **una differenza molto significativa**: con l'uguale avremmo cercato la parola precisa, invece con LIKE eseguiamo una ricerca **"non precisa"**. I due caratteri di percentuale prima e dopo la variabile della parola cercata servono infatti per dare una **"connotazione"** all'operatore LIKE, nel senso che immettendolo **prima della variabile**, dichiariamo che **potrebbero esserci altri caratteri prima della parola che abbiamo cercato**, e invece immettendolo dopo non facciamo altro che dichiarare che potrebbero esserci altri caratteri **subito dopo**.

Ma vediamo un esempio concreto. Supponiamo di avere tre post, il primo che si intitola "Ciao", il secondo che si intitola "Ciao a tutti!" e il terzo "Un ciao ai miei amici!". E supponiamo anche di avere alcune **query di ricerca** (per semplicità operiamo soltanto sul titolo):

```
$query = “SELECT * FROM Post WHERE Titolo = ‘$parolacercata’”;

$query = “SELECT * FROM Post WHERE Titolo LIKE ‘$parolacercata’”;

$query = “SELECT * FROM Post WHERE Titolo LIKE ‘%$parolacercata’”;

$query = “SELECT * FROM Post WHERE Titolo LIKE ‘$parolacercata%’”;

$query = “SELECT * FROM Post WHERE Titolo LIKE ‘%$parolacercata%’”;
```

Supponiamo adesso di cercare la parola “ciao”. Vediamo i risultati per i cinque casi che abbiamo elencato sopra:

. *Caso 1*: viene trovato **soltanto il post intitolato “Ciao”**, perché abbiamo chiesto informazioni precise, ovvero i post il cui titolo è **esattamente uguale alla parola cercata** (maiuscole e minuscole non contano perché **SQL non è case sensitive** come abbiamo avuto già modo di rimarcare);

. *Caso 2*: **è del tutto equivalente** al caso 1 (quindi **è praticamente inutile usare LIKE da solo**);

. *Caso 3*: vengono trovati i post che contengono la parola cercata **anche se ci sono caratteri che la precedono**. Nel nostro caso troviamo soltanto “Ciao”, perché è vero che il post “Un ciao ai miei amici!” ha la parola “ciao” preceduta da altri caratteri, ma è anche seguita da un’altra sequenza (“ ai miei amici”), quindi **non soddisfa i requisiti di ricerca**. Se avessimo avuto un post intitolato “Un ciao”, sarebbe stato restituito come risultato;

. *Caso 4*: **il contrario del precedente**, vengono trovati “Ciao” e “Ciao a tutti!”. Abbiamo infatti cercato i record dove il campo titolo contiene la parola cercata ma **può anche contenere una stringa di caratteri dopo**. “Ciao” e “Ciao a tutti!” soddisfano questa condizione;

. *Caso 5*: vengono trovati tutti e tre i post, dal momento che abbiamo detto di estrarre i record il cui campo titolo contiene la parola cercata, che può essere **o preceduta o seguita da altre sequenze di caratteri**.

Possiamo poi anche fare **ricerche “negative”**:

```
$query = “SELECT * FROM Post WHERE Titolo NOT LIKE ‘%$parolacercata%’”;
```

In questo caso, anteponendo quindi **la parola NOT all’operatore LIKE**, la nostra interrogazione fa in modo che vengano estratti i post il cui titolo **non contiene la parola cercata**, preceduta o seguita da altre stringhe di caratteri. E poi abbiamo anche un altro carattere speciale, l’**underscore**:

```
$query = “SELECT * FROM Post WHERE Titolo LIKE ‘_$parolacercata_’”;
```

In questo caso la nostra interrogazione estrae tutti i post che contengono la parola cercata, **preceduta o seguita da un solo carattere**: pertanto, se il carattere di percentuale equivale a **“qualsiasi numero di caratteri”**, l’underscore equivale a **“un solo carattere”**.

Potremmo però voler cercare una parola che sia **effettivamente preceduta da un underscore**, e quindi non volerlo considerare come carattere speciale. In questo caso dovremo scrivere questo:

```
$query = “SELECT * FROM post WHERE Titolo NOT LIKE ‘_$parolacercata’ ESCAPE”;
```

Aggiungendo **la specifica ESCAPE** facciamo capire che quello che precede la variabile cercata è un vero underscore e non un carattere speciale.

Bene, una volta inviata la nostra interrogazione possiamo procedere con la stampa:

```
while ($record = mysql_fetch_array($risultato, MYSQL_ASSOC))

{

   echo $record[‘Titolo’]; echo “<br />”;

}
```

In questo caso stampiamo soltanto il titolo, ma possiamo decidere a piacere come far apparire all’utente il risultato della nostra ricerca. Quindi, per terminare il capitolo, riepiloghiamo tutto il codice della pagina “ricerca.php”:

```
if(isset($_POST[‘vai’]))

{

   echo “Ecco i risultati della tua ricerca:<br /><br /> “;

   $parolacercata = $_POST[‘cerca’];

   $query = “SELECT * FROM post WHERE Titolo NOT LIKE ‘%$parolacercata%’”;

   $risultato = mysql_query($query);

   while ($record = mysql_fetch_array($risultato, MYSQL_ASSOC))

   {

      echo $record[‘Titolo’]; echo “<br />”;

   }

}

else

{

}
```

26. Le sessioni e i cookie

Con questo capitolo vedremo in che modo, con PHP, è possibile **conservare informazioni durante la navigazione**: si tratta di un aspetto **molto utile** soprattutto se è necessario costruire un sito web che debba consentire ai propri utenti di **fare il login** e di navigare le pagine in qualità di **utenti registrati**. Per fare questo abbiamo **due strumenti a nostra disposizione**: il primo è costituito dalle **sessioni**, il secondo dai **cookie**.

Iniziamo dalle **sessioni**: dobbiamo immaginare le sessioni come "contenitori" in cui possiamo inserire **diverse informazioni**. Queste informazioni verranno **memorizzate sul server** e dureranno il tempo di una… "sessione", appunto, perché quando il browser verrà chiuso tutte queste informazioni **saranno cancellate**.

PHP mette a disposizione di chiunque voglia creare una sessione **la variabile superglobale $_SESSION**, che funziona proprio come tutte le altre variabili superglobali di cui abbiamo già parlato nei capitoli precedenti. Prima di tutto però c'è **un'operazione preliminare da compiere**, ovvero **avviare la sessione**. Lo si fa con una sezione apposita, *session_start*, che deve obbligatoriamente essere inserita **prima di qualsiasi tipo di output** all'interno del nostro file PHP:

```
<?php



session_start();

// codice…



?>
```

La funzione *session_start* (la cui documentazione ufficiale si può trovare all'indirizzo http://php.net/manual/en/function.session-start.php) non solo permette di **creare una sessione**, ma permette anche di **recuperare dati** da una sessione **creata in un'altra pagina**. Ma come si fa a **creare una sessione**? Basta semplicemente **utilizzare la variabile superglobale $_SESSION** e **associare a essa le informazioni**:

```
$_SESSION['nome'] = "Federico";

$_SESSION['id'] = "1";
```

```
$_SESSION[‘livello’] = “3”;
```

Ovviamente queste informazioni, nel caso di un utente che si collega al nostro sito, si possono agevolmente **recuperare da un database** (nell’esempio ometterò, per semplicità, la query, perché parleremo **nel prossimo capitolo** di come recuperare **in modo corretto** e allo stesso tempo semplice i dati immessi da un utente per fare un login):

```
$query = // ipotetica select per verificare i dati inseriti dall’utente

$result = mysql_query($query);

$numero_record = mysql_num_rows($result);



if($numero_record == 1)

{



  while($row = mysql_fetch_array($result, MYSQL_ASSOC))

  {



    $_SESSION[‘id’] = $row[‘id’];

    $_SESSION[‘nome’] = $row[‘nome’];

    $_SESSION[‘livello’] = $row[‘livello’];



  }



echo “Login effettuato con successo!”;



}
```

In tutte le pagine che vorremo riservare agli utenti registrati, dovremo inserire **prima di**

qualsiasi altra cosa la funzione *session_start*: se omettessimo la funzione, la pagina non potrebbe ricevere le informazioni delle sessioni. Se per esempio con un blocco if controlliamo che l'utente sia loggato (per esempio verificando, tramite *isset*, che sia impostato il **livello dell'utente** a sua volta memorizzato nella variabile superglobale $_SESSION) ma allo stesso tempo **dimentichiamo** di inserire la funzione *session_start*, la conseguenza sarà che la pagina non conoscerà le informazioni memorizzate nella sessione e la nostra applicazione si comporterà come se l'utente **non si fosse mai loggato al sito**. Attenzione anche a **non dichiarare due volte la funzione** (spesso, includendo file nella pagina, può capitare di farlo), perché *session_start* può essere dichiarata **una volta soltanto**.

Le sessioni possono essere poi **azzerate o eliminate**: per fare questo è possibile utilizzare le funzioni *session_unset* e *session_destroy*. La differenza consiste nel fatto che la prima funzione **elimina soltanto i dati** (senza cancellare **gli elementi dell'array** che abbiamo dichiarato in precedenza), la seconda invece **elimina dati e array**. Pertanto in seguito a *session_destroy* **sarà necessario dichiarare nuovamente** *session_start*. Sono utili in funzioni di **logout** e si invocano in modo molto semplice:

```
session_unset();

session_destroy();
```

Parliamo adesso dei **cookie**: molti di voi ne avranno già sentito parlare. I cookie (letteralmente "biscotti") sono **informazioni** che vengono inviate **dal server al client**: e proprio qui sta la prima differenza rispetto alle sessioni, perché queste ultime **rimangono sul server**. Un'altra differenza consiste nel fatto che la sessione viene **cancellata** nel momento in cui il browser **viene chiuso**: il cookie invece **persiste fino alla sua scadenza**, impostata quando il cookie viene creato. Data la natura dei cookie, questi ci torneranno utili se vogliamo creare un modulo di login che si **"ricordi"** dell'utente **una volta chiuso il browser**, per evitargli magari di immettere nuovamente nome e password al collegamento successivo. Un altro esempio può essere dato da **un sito multilingua**: se il nostro sito accoglie l'utente in italiano, ma quest'ultimo preferisce l'inglese, potrebbe rendersi necessario un cookie che **si ricordi dell'impostazione settata dall'utente** (ovvero "sito in lingua inglese") **ogni volta che si collega**.

Per creare un cookie esiste un'apposita funzione, *setcookie*, che riceve tre parametri principali: **il nome del cookie**, le **informazioni** e la **durata**:

```
setcookie("nome", "Federico", time()+3600);
```

La durata si imposta utilizzando la funzione *time*: quest'ultima restituisce **il numero di secondi trascorsi dal 1 gennaio 1970 alle ore 00:00:00**. Si tratta della data di inizio del tempo calcolato sui **sistemi Unix** ed è, come già detto, **espresso in secondi**: si intuisce quindi che al valore restituito da time bisognerà aggiungere **un numero di secondi equivalente al periodo di durata** del nostro cookie. Nel caso dell'esempio, 3600 secondi: significa che il cookie **avrà la durata di un'ora** (formata da 3600 secondi). Si intuisce quindi che è possibile settare una durata qualsiasi per il nostro cookie, **anche di anni**!

```
setcookie("nome", "Federico", time()+3600*24); //un giorno

setcookie("nome", "Federico", time()+3600*24*30); // un mese

setcookie("nome", "Federico", time()+3600*24*365); // un anno
```

Se invece la scadenza non viene impostata, il cookie scadrà semplicemente **alla fine della sessione**. La documentazione ufficiale della funzione time è disponibile all'indirizzo http://php.net/manual/en/function.time.php.

Quello che abbiamo visto sopra è **l'esempio più tipico**, ma in realtà si possono estendere a **sette** i parametri che la funzione setcookie può ricevere, perché oltre a nome, valore e scadenza possiamo specificare anche **percorso** (ovvero la directory per cui è valido il cookie), il **dominio** (come per il percorso, ma si estende a tutto il dominio), la **sicurezza** (che se settata a *true* fa in modo che il cookie venga trasmesso solo attraverso protocollo sicuro https) e **httponly** (funziona come la sicurezza: se settato a *true*, fa in modo che il cookie venga trasmesso solo tramite protocollo http, e di conseguenza non sarà accessibile a linguaggi di scripting come JavaScript). Per esempio:

```
setcookie("nome", "Federico", time()+3600, "/prova/", "losnaweb.com", true, true);
```

In questo caso il cookie è valido solo per la cartella "prova" del sito "losnaweb.com" (che non va scritto con "www" anteposto), si trasmette solo in presenza di una connessione sicura ed è accessibile solo al protocollo http.

```
setcookie("nome", "Federico", time()+3600, "/");
```

In questo caso invece abbiamo creato, attraverso la formula "/", un cookie che è valido per tutte le cartelle del sito.

Per recuperare il valore di un cookie che abbiamo impostato in precedenza, sarà necessario utilizzare **la variabile superglobale $_COOKIE**, e il nome dell'elemento dell'array non sarà altro che **il primo valore** che abbiamo passato **come parametro** alla funzione *setcookie*:

```
echo $_COOKIE['nome']; // stampa "Federico"
```

È ovvio che possiamo anche inizializzare variabili a cui abbinare i valori delle nostre sessioni o dei nostri cookie:

```
$nome = $_SESSION['nome'];

$cognome = $_COOKIE['cognome'];
```

Per poi **utilizzarle comodamente** all'interno del nostro codice, magari su blocchi condizionali. Tipicamente, ci si assicura che prima le variabili di sessione e i cookie siano settati tramite funzione *isset*, onde evitare di veder comparire *warning* sul sito nel caso in cui sessioni e cookie **non siano ancora stati scritti** (per esempio quando l'utente si è appena collegato al sito **e non ha ancora fatto login**).

Abbiamo visto prima come **eliminare una sessione**. I cookie **non hanno una specifica funzione** per eliminarli, ma per farlo basta semplicemente utilizzare, di nuovo, la funzione *setcookie* (a proposito, la pagina di documentazione ufficiale è http://php.net/manual/en/function.setcookie.php) impostando però, come scadenza, **un tempo negativo**:

```
setcookie("nome", "Federico", time()-3600);
```

E così facendo **avremo eliminato il nostro cookie**. Sempre la solita funzione si utilizza se vogliamo **modificare** i valori di un cookie: basterà semplicemente invocare setcookie con i nuovi parametri.

27. La sicurezza in PHP: XSS e SQL Injection

Iniziamo in questo capitolo una breve panoramica su due dei **problemi di sicurezza più comuni** degli script PHP. Partiamo dal cosiddetto **“Cross-site Scripting”**, noto anche come **XSS**: si tratta di un problema, molto diffuso, che riguarda **la sicurezza dei form** e in particolare i form che **non applicano controlli sui campi**. Il malintenzionato può, tramite il form, far eseguire un codice JavaScript che magari **rimanda a un suo sito** dove è contenuto **uno script dannoso**. A proposito: ricordo a tutti quelli che leggono che alcune delle cose che sto per illustrare, se messe in pratica, **costituiscono un reato penalmente perseguibile**. **Ve ne parlo solo affinché vi possiate difendere**.

Tornando al nostro XSS, prendiamo un semplicissimo form per inserire un messaggio:

```
<form action=“xss.php” method=“get”>

Nome:<br />

<input type=“text” name=“nome” /><br />

Messaggio: <br />

<textarea rows=“10” cols=“20” name=“messaggio”></textarea><br />

<input type=“submit” name=“invia” value=“Invia!” /><br />

</form>
```

E supponiamo che il codice della pagina xss.php sia strutturato in questo modo:

```
<?php



echo “Nome: {$_GET[‘nome’]}”;

echo “<br /><br />Messaggio: <br /><br />”;

echo stripslashes($_GET[‘messaggio’]);



?>
```

Bene, questo è **un codice del tutto insicuro** perché **non ci sono controlli di sicurezza**. Nel campo “messaggio” l’hacker può **facilmente inserire un codice JavaScript** di questo tipo:

```
<script language=“text/javascript”>document.location.href=“http://www.losnaweb.com”;</script>
```

Dove ovviamente al posto di losnaweb.com c’è **un sito malvagio**. Attraverso la stampa con *echo*, il codice **viene inserito nella pagina xss.php e quindi eseguito**. E ovviamente, date le tante cose che si possono fare con JavaScript, il reindirizzamento non è che **una possibilità**: il malintenzionato può infatti anche **installare script dannosi direttamente all’interno del sito**. E questo tipo di attacco funziona sia che il nostro form abbia metodo **get**, sia che abbia metodo **post**. Tipicamente, gli script XSS vengono utilizzati per **rubare cookie** ai malcapitati.

Difendersi da questo tipo di attacchi **è veramente molto semplice**: bisogna **solo tenerlo a mente**! Basterà fare in modo che i **caratteri riservati del linguaggio HTML vengano sostituiti con le relative entità** (per esempio la parentesi uncinata aperta con <) o che le etichette HTML vengano del tutto cancellate.

Per fare questo abbiamo tre funzioni: *htmlspecialchars*, *htmlentities* e *strip_tags*. **Funzionano tutte allo stesso modo**: ricevono come parametro **la stringa per la quale dobbiamo sostituire i caratteri**. Ecco quindi qual è il codice corretto e sicuro da utilizzare nella pagina xss.php:

```
<?php



echo “<br /><br />Messaggio: <br /><br />”;

echo htmlspecialchars(stripslashes($_GET[‘messaggio’]));



?>
```

Oppure

```
echo htmlentities(stripslashes($_GET[‘messaggio’]));
```

O ancora

```
echo strip_tags(stripslashes($_GET[‘messaggio’]));
```

Ci sono ovviamente **alcune differenze tra le tre funzioni**. La differenza tra *htmlspecialchars* e *htmlentities* consiste nel fatto che la prima converte in entità **solo i caratteri riservati di HTML**, mentre la seconda converte **anche i caratteri speciali** (per esempio, se dovessimo scrivere in tedesco, la o con la dieresi – ö – sarebbe convertita in ö). La funzione *strip_tags* invece **elimina del tutto le etichette HTML**. In ogni caso, il codice immesso dal malintenzionato verrebbe **stampato sullo schermo e non eseguit**o (anzi, nel caso di *strip_tags* **le tag verrebbero del tutto eliminate**). E **ricordate sempre**: uno dei migliori metodi per difendersi dagli attacchi è **filtrare tutto ciò che arriva in input sul nostro sito**. In questo caso abbiamo visto come filtrare attraverso le tre funzioni di cui abbiamo parlato: potete trovare le documentazioni ufficiali delle funzioni a queste pagine: http://www.php.net/manual/en/function.htmlspecialchars.php (*htmlspecialchars*), http://php.net/manual/en/function.htmlentities.php (*htmlentities*), http://it.php.net/manual/en/function.strip-tags.php (*strip_tags*).

Il secondo problema di vulnerabilità che vediamo in questo capitolo è la cosiddetta **SQL Injection** (iniezione di SQL). Anche in questo caso l’attacco prende di mira **form scritti male** per eseguire **un codice SQL malevolo**, e adesso vedremo nel dettaglio in cosa consiste questa pratica e **perché è tanto pericolosa**. Prendiamo per esempio questo semplice form per connettersi a un sito:

```
<form action=“login.php?step=2” method=“post”>

Nickname:<br />

<input type=“text” name=“nickname” /><br />

Password: <br />

<input type=“password” name=“psw” /><br />

<input type=“submit” name=“invia” value=“Invia!” /><br />

</form>
```

In questo caso il nostro form, invece di rimandare a una nuova pagina, rimanda alla stessa ma con l’aggiunta della query string **?step=2**, in modo da scrivere **tutto nella stessa pagina**. L’azione del form andrà quindi scritta in un blocco if di questo tipo:

```
if (isset($_GET[‘step’]) && $_GET[‘step’] == 2)

{

   // azione

}

else

{

}
```

Ipotizziamo anche che il gestore del sito o del blog sia stato così sprovveduto da aver fatto in modo che le password **vengano inserite in chiaro nel database** (quindi senza criptarle attraverso il form di registrazione). Ora prendiamo il pessimo script che esegue il controllo sui dati immessi dall’utente:

```
$query = “SELECT * FROM autori WHERE Nickname = ‘{$_POST[‘nickname’]}’ AND Password = ‘{$_POST[‘psw’]}’”;

$risultato = mysql_query($query);

if (mysql_num_rows($risultato) == ‘1’)

{

   echo “Benvenuto, utente autenticato”;

   // codice che mostra l’eventuale pagina personale dell’utente

}

else

{

   echo “Spiacente, password errata”;

}
```

Chiariamo subito la funzione nuova, *mysql_num_rows*: restituisce **il numero dei record trovati dalla query** (nel nostro caso uno, anche perché si suppone che **gli username siano univoci**) e riceve come parametro il risultato della query. Qua si trova la

documentazione ufficiale della funzione: http://php.net/manual/en/function.mysql-num-rows.php. Pertanto se la funzione dà come risultato 1 (e quindi trova il record), **l'utente sarà loggato**, in caso contrario ci sarà un messaggio che informerà l'utente di **aver inserito una password errata**.

Questo codice **è estremamente vulnerabile**, soprattutto nel caso in cui l'impostazione *magic quotes gpc* di PHP sia disattivata: in locale potete controllarlo su WAMP cliccando sull'icona nella barra delle applicazioni, quindi andare alla voce PHP e ancora andare su Impostazioni PHP. Se la voce "magic quotes gpc" è spuntata significa che è attiva e avete **maggiori protezioni contro le SQL Injection**. Ci sono però tanti provider che non attivano le *magic quotes* sui loro server. **Ma cosa sono le magic quotes**? Si tratta di un processo che **aggiunge in automatico i caratteri di escape alle stringhe**, ma dobbiamo sapere che a partire dalla versione 5.3.0 di PHP, **magic quotes è deprecato** e nella versione 6 **addirittura non c'è più**.

Vediamo quindi cosa può fare il malintenzionato. Nella casella nickname può inserire il nome "Federico", e fin qui niente di strano. Ma nella casella password può tranquillamente inserire questa stringa:

' OR Nickname='Federico

Così facendo, il nostro codice eseguirebbe questa query:

```
$query = "SELECT * FROM autori WHERE Nickname = 'Federico' AND Password =

" OR Nickname='Federico'";
```

La clausola OR inserita dal malintenzionato **vanificherà il controllo sulla password**: questo perché la query strutturata in tal modo dice di selezionare tutti i record della tabella autori il cui nickname sia Federico e la cui password sia vuota **oppure**, **in alternativa a quest'ultima clausola**, il cui nickname sia Federico. Dal momento che la seconda clausola è soddisfatta il malintenzionato **riuscirà a entrare agevolmente** al posto di Federico all'interno del sito, spacciandosi per lui e magari rubando i suoi dati… o insomma facendo tutto ciò che di cattivo si può fare entrando nel profilo di un'altra persona.

La prima operazione da fare è **criptare la password** attraverso la funzione *md5*: nel database **non dovrà essere inserita in chiaro** ma andrà fatta passare attraverso questa funzione (in un'ipotetica query con comando INSERT INTO metteremo tra i VALUES 'md5($_POST['password']'). **MD5** è un **algoritmo crittografico di hashing** che fu

realizzato nel 1991 da **Ronald Linn Rivest**, un crittografo statunitense: l'algoritmo **trasforma una stringa in un'altra stringa a 128 bit**, e **non è possibile risalire alla stringa originale** (se non andando… a tentativi). La funzione md5 di PHP riceve come parametro semplicemente la stringa da criptare e la documentazione ufficiale si trova qui: http://it.php.net/manual/en/function.md5.php.

Questo è solo uno dei tanti modi per criptare una password: parliamo di questo perché **è il più semplice da utilizzare**, ma c'è chi usa anche **sistemi più sofisticati**. Il nostro codice pertanto dovrà presentarsi in questa forma:

```
$query = "SELECT * FROM autori WHERE Nickname = '{$_POST['nickname']}' AND Password = '".
md5($_POST['psw']). "'";
```

Dove i punti che separano md5($_POST['psw']) dal resto del codice sono un modo per concatenare valori all'interno di una variabile (le virgole non funzionano).

In questo caso ci salviamo dal malintenzionato che immette i dati di cui sopra, **ma siamo ancora vulnerabili**. Questo perché l'hacker potrebbe scrivere nella casella del nickname **questa stringa cattivissima**:

Federico' —

Dove dopo i due trattini c'è uno spazio. E magari può anche lasciare in bianco il campo della password. In tal modo la query diventerebbe questa:

```
$query = "SELECT * FROM autori WHERE Nickname = 'Federico' — AND Password =

"";
```

I due trattini nel linguaggio SQL indicano i commenti: pertanto tutto ciò che viene dopo i due trattini sarà ignorato. Capite quindi la pericolosità della cosa malgrado il controllo tramite algoritmo md5. Sarà un giochetto per il malintenzionato riuscire a entrare nel sito e magari a carpire tutta la lista degli utenti registrati o a far di peggio: pensate se potesse entrare con l'account dell'amministratore.

Qual è pertanto il modo più sicuro per difendersi da questo tipo di attacco? Non dovremo far altro che aggiungere una funzione per controllare i dati immessi dall'utente, e si tratta della funzione mysql_real_escape_string, che inserisce i giusti caratteri di escape prima dei caratteri "pericolosi". Riceve come parametro semplicemente la stringa a cui

aggiungere i parametri di escape. Quindi la stringa della nostra query dovrà assumere questa forma per evitare attacchi:

```
$query = “SELECT * FROM autori WHERE Nickname = ’”.
mysql_real_escape_string($_POST[‘nickname’]) .”’ AND Password = ’”.
mysql_real_escape_string($_POST[‘psw’]). ”’”;
```

Riusciremo così a rendere inoffensivo l’attacco del malintenzionato, perché l’ultima stringa che abbiamo visto risulterà invece in questo modo:

$query = “SELECT * FROM autori WHERE Nickname = ‘Federico' — AND Password = ”’”;

Questa funzione va utilizzata solo nel caso in cui magic quotes gpc sia disattivo. Per essere però del tutto sicuri utilizziamo la funzione get_magic_quotes_gpc, che controlla la configurazione delle magic quotes (non riceve alcunché come parametro) e utilizzarla in un ciclo if:

```
if(get_magic_quotes_gpc())

{

  $query = “SELECT * FROM autori WHERE Nickname = ’”. $_POST[‘nickname’] .”’ AND Password
= ’”. $_POST[‘psw’]. ”’”;

}

else

{

  $query = “SELECT * FROM autori WHERE Nickname = ’”.
mysql_real_escape_string($_POST[‘nickname’]) .”’ AND Password = ’”.
mysql_real_escape_string($_POST[‘psw’]). ”’”;

}
```

Controlliamo con il ciclo che il processo magic quotes gpc sia attivo, e in caso affermativo creiamo la nostra query senza la funzione mysql_real_escape_string, e in caso contrario ovviamente la aggiungiamo. Trovate la documentazione ufficiale di queste due funzioni qui http://it.php.net/manual/en/function.mysql-real-escape-string.php (mysql_real_escape_string) e qui http://php.net/manual/en/function.get-magic-quotes-

gpc.php (get_magic_quotes_gpc).

28. La sicurezza in PHP: Session hijacking

Terminiamo con questo capitolo la breve panoramica sui più comuni problemi di sicurezza con alcuni cenni sul **session hijacking**, che consiste nel **“furto dell’identità”** di un utente che si collega a un sito web.

Che cosa fa il malintenzionato che sottopone l’utente di un sito a session hijacking? Non fa altro che **rubare l’identificativo di sessione dell’utente** (ogni sessione **ha un proprio id**) per poi loggarsi al sito **spacciandosi per l’utente** a cui ha rubato le credenziali. Uno dei metodi più comuni per rubare le credenziali è **sfruttare il cross-site scripting**: abbiamo però già parlato nel capitolo precedente di come difenderci da questo tipo di attacchi, quindi vedremo come coprirci anche da un furto ottenuto tramite **“session fixation”**: con la **session fixation**, i dati verranno rubati tramite **una sessione creata *ad hoc*** dal malintenzionato.

Certo, affinché si verifichi una session fixation **devono essere soddisfatte due condizioni**: **codice vulnerabile** e **utente sprovveduto**. Nell capitolo sulle sessioni non abbiamo parlato del modo in cui l’id di sessione **viene propagato**: ciò può avvenire **tramite un cookie** che viene appositamente creato per contenere l’id (cookie che si estinguerà alla chiusura del browser) oppure, nel caso in cui i cookie siano stati disabilitati, **tramite metodi get e post**. Il ladro di sessioni può convincere l’utente del nostro sito (magari tramite una mail o con un post su un blog) a collegarsi a un indirizzo simile a questo: www.nostrosito.it/login.php?PHPSESSID=12345 dove, **attraverso la query string** e ipotizzando che l’id di sessione venga **propagato tramite get**, decide il numero di sessione.

Per comprendere meglio il meccanismo di un attacco session fixation proviamo a simularne uno (ma, come detto nel capitolo precedente, solo affinché vi possiate difendere). Per prima cosa assicuriamoci che nelle impostazioni del nostro PHP sia abilitata la trasmissione dei dati di sessione **attraverso url**: clicchiamo quindi sull’icona di Wamp, andiamo sotto la voce PHP e quindi apriamo il file php.ini. Cerchiamo adesso la direttiva *session.use_trans_sid*, che è quella che consente la **tramissione degli id di sessione via url** (a proposito: i commenti sopra di essa vi diranno che **abilitarla sottopone a rischi di sicurezza**). Se è a 1 lasciamola com’è, altrimenti se è impostata a 0 dobbiamo settarla a 1. Cerchiamo poi la direttiva *session.use_only_cookies* (permette la trasmissione degli id di sessione **solo tramite cookie**) e impostiamola a 0.

A questo punto creiamo un semplice codice PHP che al nostro collegamento dà inizio a una sessione che memorizza **il numero di volte in cui visualizziamo la pagina** con il nostro browser:

```
<?php



session_start();

if (!isset($_SESSION[‘conto’]))

{

  $_SESSION[‘conto’] = 0;

} else

{

  $_SESSION[‘conto’]++;

}



echo “Numero:” . $_SESSION[‘conto’].””;

?>
```

Tutto molto semplice. Con la solita funzione *isset* controlliamo che la variabile $_SESSION[‘conto’] non sia stata inizializzata: se non esiste, la creiamo e la impostiamo a 0, se invece esiste già la incrementiamo. Terminato il blocco if stampiamo la nostra variabile.

Salvate il file chiamandolo, per esempio, “fixation.php” e inseritelo in una cartella “prova” all’interno della cartella “www” di Wamp. A questo punto aprite il vostro browser preferito (per esempio Firefox), **eliminate tutti i cookie** di localhost e collegatevi a questo indirizzo: http://localhost/prova/fixation.php?PHPSESSID=12345. Aggiornate la pagina un qualsiasi numero di volte, arriviamo per esempio a 9.

A questo punto chiudete pure il browser. Riaprite un altro browser (esempio, Chrome), magari anche su un altro computer se fate la prova su un server remoto, e provate a collegarvi allo stesso indirizzo: come potrete notare il conto non ripartirà da 0 ma partirà da dove siete rimasti prima (nel nostro caso vedremo “10”). Cosa significa questo? Abbiamo simulato **un ipotetico utente A** che utilizza Firefox, apre la pagina con il link **che gli è stato suggerito dal malintenzionato utente B** e dà inizio a una sessione. L’utente B aprirà poi il suo Chrome **con l’id di sessione da lui stesso creato** e **potrà accedere ai dati dell’utente A**. In questo caso si tratta semplicemente del numero di volte che ha aggiornato la pagina, ma questo tipo di attacco funziona tranquillamente anche su

script molto più complessi (per esempio un login a un sito web: il malintenzionato B potrà accedere indisturbato a tutti i dati personali dell'utente A) che però **hanno seri problemi di sicurezza**.

Come difendersi da questo attacco? Il metodo più semplice è **rigenerare spesso l'id di sessione**, in modo da rendere inefficaci gli attacchi: questo si può fare agevolmente tramite l'apposita funzione *session_regenerate_id*, che si applica in modo molto semplice:

```
session_regenerate_id();
```

E in genere va fatto **dopo aver dato il via alla sessione** con *session_start*. La documentazione ufficiale di questa funzione si trova all'indirizzo http://php.net/manual/en/function.session-regenerate-id.php. Inoltre sarebbe bene che le direttive di cui sopra, *session.use_trans_sid* e *session.use_only_cookies*, fossero impostate rispettivamente a 0 e 1. Comunque l'utilizzo di *session_regenerate_id* è di per sé già sufficiente per difendersi. È quindi bene **rigenerare spesso**, anche **prima che l'utente abbia fatto login** (inserendo quindi la funzione nella pagina di login, per esempio).

Infine potremmo eseguire **un controllo aggiuntivo sullo user agent dell'utente** (perché niente impedisce che **uno stesso id di sessione**, come abbiamo visto, **possa essere utilizzato da due utenti diversi**): questo consente anche di avere una difesa contro attacchi di tipo **"session sidejacking"**. Attraverso questo tipo di attacco, comunque difficile da mettere in atto, il malintenzionato cerca di **intercettare l'id di sessione attraverso i pacchetti di rete** che vengono scambiati tra il server e il browser dell'utente. Per ovviare a questa minaccia può essere utile fare in modo che **l'accesso venga impedito** nel caso in cui lo *user agent* di chi richiede l'accesso al nostro sito **non corrisponda** a quello dell'utente che ha effettuato il login.

Per fare questo abbiamo bisogno della variabile superglobale **$_SESSION**: dovremo associare a essa il valore dell'elemento **HTTP_USER_AGENT** della variabile superglobale **$_SERVER**, **meglio se codificato tramite funzione md5**. Il tutto nella pagina o nella funzione che consente il login:

```
$_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
```

In questo modo abbiamo creato un elemento della sessione chiamato HTTP_USER_AGENT a cui abbiamo associato nient'altro che il nostro user agent criptato con md5. A questo punto, per eseguire il controllo, potremmo utilizzare un codice

simile a questo:

```
if (isset($_SESSION[‘HTTP_USER_AGENT’]))

{

  if ($_SESSION[‘HTTP_USER_AGENT’] != md5($_SERVER[‘HTTP_USER_AGENT’]))

  {

    logout();

  }

}
```

Con questo script non facciamo altro che controllare, sempre tramite la funzione *isset*, se esiste la sessione con elemento “HTTP_USER_AGENT”: in caso affermativo, se il suo valore **è diverso dallo user agent**, sempre criptato con md5, **dell’utente che sta eseguendo lo script** (in altre parole se **chi si è loggato ha uno user agent diverso** rispetto a quello **dell’utente che sta cercando di forzare l’accesso al sito**), rimandiamo l’utente a una ipotetica funzione logout **che distruggerà la sessione**.

29. Le espressioni regolari

Siamo arrivati all'ultimo capitolo di **questo corso base al linguaggio PHP**, e per concludere affrontiamo un argomento di sicuro **non semplice** ma che cercheremo di trattare in modo tale da fornire alcuni strumenti utili per la sua comprensione: stiamo parlando delle **espressioni regolari**. In un solo capitolo **non riusciremo** a trattare l'argomento in modo completo al 100% ma con le nozioni che avremo alla fine potremo **eseguire diverse operazioni**.

Per iniziare possiamo dire che cosa sono e a che cosa servono le espressioni regolari. Si tratta di particolari **costrutti** che si usano per verificare se all'interno di una stringa (e quando in PHP si parla di stringa si intende anche **un testo lungo e complesso**) è presente una certa sottostringa. Potreste pensare che in realtà non ci sono novità perché abbiamo già visto, nel capitolo dedicato alle funzioni sulle stringhe, come **cercare e sostituire sottostringhe**. In realtà le espressioni regolari **sono uno strumento molto più potente** perché non definiscono semplici sottostringhe ma, data la loro natura versatile, con **un solo *pattern*** (una **sequenza di caratteri**) possiamo cercare una **grande varietà di sottostringhe**.

Le espressioni regolari possono essere utili per moltissime situazioni: per esempio, nel capitolo in cui parlavamo dei form abbiamo fatto un cenno alle espressioni regolari perché abbiamo detto che servono per controllare se l'utente inserisce **un indirizzo email corretto**. Questo infatti potrebbe essere un utilizzo, ma potremmo usare le espressioni regolari per altre operazioni, per esempio controllare **se un dominio è stato scritto correttamente**, **evidenziare una certa occorrenza** o un insieme di occorrenze in un testo, **cercare tutte le etichette HTML** presenti in un testo, **controllare la complessità di una password** e via dicendo.

Entriamo subito nell'argomento e iniziamo a parlare di **"metacaratteri"**, ovvero di caratteri speciali inseriti all'interno di una espressione regolare e che sono dotati **di un significato ben preciso**. Per esempio la sequenza **[a-z]** indica "qualsiasi lettera minuscola compresa tra la a e la z". Se volessimo invece trovare tutte le lettere non solo minuscole ma anche maiuscole dovremmo scrivere un'espressione come questa: **[a-zA-Z]**. Avrete quindi intuito che **le parentesi quadre racchiudono un insieme di caratteri**. Vediamo quali sono i metacaratteri:

[] (parentesi quadre)	Contengono una sequenza di caratteri
() (parentesi tonde)	Contengono una precisa sottostringa
. (punto)	Qualsiasi carattere

^ (accento circonflesso)	Inizio di stringa o negazione
$ (dollaro)	Fine della stringa
| (pipe)	Operatore disgiuntivo OR
\ (backslash)	Carattere di escape

Esistono poi i **quantificatori**, che indicano **quante volte** bisogna cercare un'occorrenza all'interno della nostra stringa principale:

* (asterisco)	Zero o più occorrenze
? (punto interrogativo)	Zero o una occorrenza
+ (più)	Una o più occorrenze
{} (parentesi graffe)	Contengono il numero che rappresenta la ripetizione di un'occorrenza.

Facciamo alcuni esempi per comprendere meglio:

[a-zA-Zàèéìòù0-9]	Tutti i caratteri minuscoli e maiuscoli, le lettere accentate della lingua italiana e i numeri
[^0-9]	Un qualsiasi carattere che non sia un numero
(ciao)	La sottostringa "ciao"
t?o	La lettera o preceduta da 0 o 1 occorrenze della lettera t (se il testo è "gatto, canto, albero" ci saranno tre risultati utili e cioè la sequenza "to" di gatto, la sequenza "to" di canto e la "o" di albero)
t*o	La lettera o preceduta da 0 o più occorrenze della lettera t (per il testo precedente ci saranno tre risultati utili e cioè "tto" di gatto, "to" di canto e "o" di albero)

t+o	La lettera o preceduta da 1 o più occorrenze della lettera t (due risultati utili, “tto” di gatto e “to” di canto)
t{2}o	La lettera o preceduta da 2 occorrenze della lettera t (un risultato, “tto” di gatto)
t{3,5}o	La lettera o preceduta da un minimo di 3 fino a un massimo di 5 occorrenze della lettera t (nessun risultato)
[^t]t{1}o	Un’occorrenza della lettera t seguita dalla lettera o e preceduta da qualsiasi carattere purché non sia una t (un risultato, “to” di canto)
[a-s](to)	Qualsiasi carattere compreso tra a e s seguito dalla sottostringa “to” (un solo risultato, “to” di canto).
(2\+2)	La stringa “2+2”
.$	Qualsiasi carattere purché sia l’ultimo della stringa
(to)|(ro)	La sottostringa “to” o la sottostringa “ro” (tre risultati: “to” di gatto, “to” di canto e “ro” di albero

E così via. A questo punto abbiamo quindi gli strumenti per creare dei pattern utili… partiamo dall’esempio più semplice, ovvero **un pattern per validare una partita IVA** che come tutti sappiamo è composta da undici numeri:

^[0-9]{11}$

Non facciamo altro che creare una sequenza [0-9] **che indica un qualsiasi numero**, e la facciamo seguire subito dalla sequenza {11} che indica che il numero **deve obbligatoriamente occorrere undici volte**. L’accento circonflesso e il dollaro indicano ovviamente **l’inizio e la fine della stringa**. Se omettessimo uno dei due, **avremmo vanificato il controllo** in quanto potremmo inserire un qualsiasi numero prima o dopo la nostra partita IVA e il pattern verrebbe comunque validato perché il nostro script non saprebbe **da dove inizia o dove finisce la stringa**.

Possiamo poi creare un **pattern che controlli che un dato file sia un’immagine**:

^.+.(jpeg|jpg|png|gif)$

Abbiamo qui il punto che indica **una qualsiasi sequenza di caratteri**, seguito da un “vero” punto (per essere considerato tale, **deve essere preceduto dal carattere di escape**) e quindi dalle estensioni tipiche delle immagini “da web” (jpeg, jpg, png, gif) **tra parentesi tonde** e separate tra di loro per mezzo dell’**operatore disgiuntivo**.

Prendiamo poi, per esempio, **un controllo per validare una data**:

^[0-9]{1,2}.[0-9]{1,2}.[0-9]{4}$

In questo caso non facciamo altro che creare tre sequenze [0-9] **rispettivamente per i giorni, i mesi e gli anni**: nei primi due casi il numero può comparire **da un minimo di una a un massimo di due volte** e deve essere seguito da un punto (con cui separiamo i numeri della data… gusti personali) a sua volta preceduto da un **backslash**. Per quanto riguarda l’anno invece il numero deve essere **obbligatoriamente presente quattro volte**.

Più complesso invece è il **pattern per validare un indirizzo email**:

^[a-zA-Z0-9_.-]+@[a-zA-Z0-9_.-]+.[a-z]{2,6}$

Nella prima sequenza tra parentesi quadre inseriamo **tutti i caratteri alfanumerici insieme all’underscore, al punto e al trattino** e indichiamo che possono ricorrere **una o più volte** attraverso l’uso del segno +, a fui facciamo seguire **la chiocciola**. Questa prima sequenza rappresenta il nome utente dell’indirizzo. Dopo la chiocciola inseriamo un’altra sequenza tra parentesi quadre simile alla prima e poi inseriamo **il punto preceduto dal carattere di escape**. Abbiamo quindi **il nome del dominio**, a cui facciamo seguire l’espressione [a-z] che può ricorrere **da un minimo di 2 fino a un massimo di 6 volte**: è nient’altro che **l’estensione del dominio**.

Per alcune delle espressioni che abbiamo utilizzato esistono poi le cosiddette **“classi di caratteri”**, ovvero **caratteri che corrispondono a precise sequenze** e che possono accorciare molto il nostro lavoro. Vediamone alcune:

\w	Corrisponde a [a-zA-Z0-9_]
\W	Corrisponde a [^a-zA-Z0-9_]
\d	Corrisponde a [0-9]

\D	Corrisponde a [^0-9]
\s	Cerca gli spazi
\S	Cerca tutto ciò che non sia uno spazio
[:alnum:]	Corrisponde a [a-zA-Z0-9]
[:alpha:]	Corrisponde a [a-zA-Z]
[:lower:]	Corrisponde a [a-z]
[:upper:]	Corrisponde a [A-Z]
[:punct:]	Cerca i caratteri di punteggiatura

Negli ultimi cinque casi, le parentesi quadre sono parte integrante della sequenza, quindi se vogliamo cercare i segni di interpunzione nella nostra stringa il pattern dovrà essere strutturato in questo modo:

[[:punct:]]

Con le classi di caratteri, le stringhe per la validazione di partite IVA e indirizzi email, tanto per prendere due degli esempi che abbiamo fatto in precedenza, potrebbero diventare una cosa di questo tipo:

^\d{11}$

^\S+@[\w.-]+.[a-z]{2,6}$

Ma come fare per **eseguire il controllo nel nostro script PHP**? Abbiamo bisogno di una funzione, *preg_match*, che riceve come parametri **il pattern e la stringa da controllare** e nel caso in cui nel testo ci sia almeno una coincidenza con il pattern, la funzione restituirà come risultato 1 (true), e in caso contrario il risultato sarà 0 (false).

Per esempio, poniamo il caso di uno script che controlla se un utente, attraverso un form,

ha caricato un'immagine con estensione corretta (il nome dell'immagine sarà stato precedentemente memorizzato in una variabile $immagine).

```
if (preg_match('/^.+.(jpeg|jpg|png|gif)$/', $immagine))

{

   echo "Grazie per aver caricato l'immagine!";

}

else

{

   echo "L'immagine ha un'estensione non valida";

}
```

I due slash inseriti dopo gli apici del primo parametro della funzione sono **i delimitatori**: se non inseriti, **lo script restituirà un errore**. Il meccanismo della funzione è molto semplice: il blocco if controlla che il risultato sia true e in caso affermativo stampa la scritta "Grazie per aver caricato l'immagine!", mentre in caso contrario la scritta mostrata all'utente sarà "L'immagine ha un'estensione non valida".

La funzione può essere usata anche, ovviamente, **nel modo inverso**, al negativo, facendo precedere la funzione **dal punto esclamativo che indica negazione**:

```
if (!preg_match('/^[a-zA-Z0-9_.-]+@[a-zA-Z0-9_.-]+.[a-z]{2,6}$/', $indirizzoemail))

{

   echo "Indirizzo email non valido";

}

else

{

   echo "Indirizzo email valido!";

}
```

In questo caso **abbiamo controllato un indirizzo email**. La documentazione ufficiale della funzione *preg_match* si può trovare all'indirizzo http://php.net/manual/en/function.preg-match.php.

Abbiamo detto poi che attraverso le espressioni regolari possiamo poi anche fare **sostituzioni all'interno di un testo**. Questa operazione si può eseguire attraverso l'utilizzo della funzione *preg_replace* che riceve **tre parametri**: l'espressione regolare che definisce **le sequenze da sostituire**, **la sottostringa sostituta** e **la stringa all'interno della quale compiere la sostituzione**. Con un esempio banale:

```
$testo = "Cane, gatto, topo, cavallo, pecora, mucca";

echo preg_replace ("/[aeiou]/", "*", $testo);
```

Attraverso l'utilizzo della funzione *preg_replace* (il cui risultato sarà direttamente stampato attraverso *echo*) cerchiamo nella stringa $testo tutte le volte in cui **occorre una vocale** tramite la sequenza **[aeiou]**, e **sostituiamo ogni vocale con un asterisco**. Questa funzione si presta a diversi utilizzi… per esempio, stiamo costruendo un blog e vogliamo censurare le parolacce dei nostri commentatori (farò un esempio… senza parolacce ma che funziona allo stesso modo).

```
$testo = "Il gatto mangia il topo";

echo preg_replace ("/(gatto|topo)/", "***", $testo);
```

In questo caso la parola "gatto" e la parola "topo" vengono sostituite dalla sequenza ***. Possiamo poi adoperare la funzione anche per **sostituire etichette HTML**:

```
$testo = "<strong>Ciao</strong>!!!!!";

$testo = preg_replace("/(<strong>)/", "<em>", $testo);

echo preg_replace("/(<\/strong>)/", "</em>", $testo);
```

In questo caso sostituiamo **tutte le parole marcate in grassetto con parole marcate in corsivo**. Questi non sono che alcuni esempi, e **la versatilità delle espressioni regolari** dà modo a ognuno di **sfruttarle al meglio per le proprie esigenze**. Per chiunque fosse interessato all'argomento, consiglio di approfondire facendo una ricerca sul web: si trovano davvero molti siti che sapranno spiegare meglio di me e in modo più completo

l’utilizzo di questo affascinante strumento.

Soluzioni degli esercizi

Capitolo 8

Prima parte: form per la prenotazione

```
<form action="esercizio1parte2.php" method="post">

Nome e cognome:<br />
<input type="text" name="nome" /><br />

Numero notti: <br />
<input type="text" name="notti" /><br />

Data di arrivo (gg/mm/aaaa):<br />
<input type="text" name="data" /><br />

Tipo di camera: <br />
<input type="text" name="camera" /><br />

Colazione: <br />
Sì <input type="radio" name="colazione" value="Con colazione" /><br />
No <input type="radio" name="colazione" value="Senza colazione" /><br /><br />

<input type="submit" name="invia" value="Invia!" /><br />
</form>
```

Seconda parte: riepilogo dati

```
<?php

echo "Salve {$_POST['nome']}, questo è il riepilogo dei suoi dati:<br /><br />";

echo "Titolare della prenotazione: {$_POST['nome']}<br />";

echo "Numero notti: {$_POST['notti']}<br />";

echo "Data di arrivo: {$_POST['data']}<br />";

echo "Tipo di camera: "; echo stripslashes($_POST['camera']); echo "<br />";
```

```
echo “{$_POST[‘colazione’]}<br />”;

?>
```

Capitolo 9

Prima parte: form per la richiesta

```
<?php

echo ‘

<form action=“esercizio2parte2.php” method=“post”>

Di lusso o economico?<br />

Di lusso <input type=“radio” name=“fascia” value=“l” /><br />

Economico <input type=“radio” name=“fascia” value=“e” /><br /><br />

Con piscina o senza?<br />

Con <input type=“radio” name=“piscina” value=“c” /><br />

Senza <input type=“radio” name=“piscina” value=“s” /><br /><br />

<input type=“submit” name=“invia” value=“Entra nel sito!” />

</form>

‘;

?>
```

Seconda parte: risposta

```
<?php

if ($_POST[‘fascia’] == ‘l’)

{

    if ($_POST[‘piscina’] == ‘c’)

    {
```

```
        echo “Il tuo hotel ideale è il Grand Hotel Villa Papera”;

    }

    else

    {

        echo “Il tuo hotel ideale è il Duck Palace Hotel”;3

    }

}

else

{

    if ($_POST[‘piscina’] == ‘c’)

    {

        echo “Il tuo hotel ideale è il Relais dei Paperi”;

    }

    else

    {

        echo “Il tuo hotel ideale è l’Hotel Papero”;

    }

}

?>
```

Capitolo 10

Prima parte: calcolatrice

```
<?php



echo “<form method=‘post’ action=‘calcolatrice-risultato.php’>

Primo numero: <input type=‘text’ name=‘num1’ /><br />
```

```
Operatore (+, -, *, /): <input type=‘text’ name=‘operatore’ /><br />

Secondo numero: <input type=‘text’ name=‘num2’ /><br /><br />

<input type=‘submit’ value=‘Risultato’ /></form>”;

?>
```

Seconda parte: risultato

```
<?php

switch($_POST[‘operatore’])
{
    case ‘+’: $risultato = $_POST[‘num1’]+$_POST[‘num2’]; break;
    case ‘-‘: $risultato = $_POST[‘num1’]-$_POST[‘num2’]; break;
    case ‘*’: $risultato = $_POST[‘num1’]*$_POST[‘num2’]; break;
    case ‘/’: $risultato = $_POST[‘num1’]/$_POST[‘num2’]; break;
    default: $risultato = ‘Hai inserito un operatore non valido!’;
}

echo $risultato;

?>
```

Capitolo 12

```
<?php

//dichiariamo un secondo elemento casuale

$casuale2 = rand(0, $ultimo_elemento);



function rotazione()

{

    // rendiamo globali le variabili

    global $immagini;

    global $ultimo_elemento;

    global $casuale;

    // stampiamo l'immagine

    echo "<img src='{$immagini[$casuale]}' alt='Immagine' />";



    //aggiungiamo il secondo elemento casuale

    global $casuale2;



    //controlliamo se il primo casuale è uguale al secondo casuale

    if ($casuale == $casuale2)

    {

        //in caso positivo, aumentiamo l'indice di 1

        $casuale2++;



        //ora controlliamo però che l'indice non sia superiore all'ultimo indice

        if ($casualeloghi2 > $ultimo_elemento)

        {

            //in questo caso, lo portiamo a 1

            $casualeloghi2 = 1;
```

```
        }

        //e a questo punto, stampiamo l'immagine

         echo "<img src='{$immagini[$casuale2]}' alt='Immagine' />";



    }



    else

    {

        //se non sono uguali, semplicemente stampiamo l'immagine

        echo "<img src='{$immagini[$casuale2]}' alt='Immagine' />";

    }



    /*nella funzione associazione, basterà aggiungere uno switch

    per la variabile globale     $casuale2.

    */

}

?>
```

Capitolo 13

Prima parte: form

```
<?php



echo "<form method='post' action='esercizio3parte2.php'><input type='text' name='num' />";

echo "<input type='submit' value='Invia' /></form>";



?>
```

Seconda parte: costruzione della torta

```
<?php


/*assegno il valore immesso dall’utente

a una variabile che chiamo $max, perché

il valore dell’utente corrisponde al

valore massimo degli strati della torta

*/


$max = $_POST[‘num’];


//stampo le candeline

echo “| | | |<br />”;


//controllo che il numero sia superiore a 0

if ($max <= 0)

{

    echo “Numero minore o uguale a zero!”;

}

//se va tutto bene, inizio il ciclo, partendo da 0

else

{

    for ($i = 0; $i<$max; $i++)

    {

        echo “–-<br />***<br />”;

    }

    /*se ho raggiunto il numero impostato dall’utente,
```

```
    stampo l'ultimo strato della torta

    */

    if ($i == $max)

    {

        echo "–-";

    }

    else

    {

    }



}



?>
```

Capitolo 14

```
<?php



//inizializziamo l'array, nel modo che preferiamo. Quello seguente è il più veloce

$classifica = array(1 => 'Tina Maze', 'Federica Brignone', 'Tessa Worley', 'Denise Karbon', 'Viktoria Rebensburg');



//stampiamo la classifica

foreach ($classifica as $posizione => $sciatrice)

{

    echo $posizione.". ".$sciatrice."<br />";

}
```

```
?>
```

Capitolo 15

Prima parte: form

```
<form action="esercizio4parte2.php" method="post" name="numero">

Numero di opere:<br />

<input type="text" name="numero" /><br />

<input type="submit" name="invia" value="Invia!" />

</form>
```

Seconda parte: stampa del nuovo form

```
<form name='opere' action='esercizio4parte3.php' method='post'>

<?php

for ($i = 0; $i<$_POST['numero']; $i++)

{

    echo "

    Opera:<br /> <input name='opera$i' type='text' /><br />

    Autore:<br /> <input name='autore$i' type='text' /><br />

    Datazione:<br /> <input name='datazione$i' type='text' /><br />

    <br />

    ";

}

?>

<input type="hidden" name='campi' value='<?php echo $_POST['numero']; ?>' />

<input type='submit' value='Invia!' />
```

Terza parte: stampa delle opere

```
<?php


for ($i=0; $i<$_POST[‘campi’]; $i++)
{

    echo “Opera: “,$_POST[“opera$i”], “<br />”;

    echo “Autore: “,$_POST[“autore$i”],”<br />”;

    echo “Datazione: “,$_POST[“datazione$i”],”<br />”;

    echo “<br />”;

}


?>
```

Capitolo 16

Prima parte: form

```
<?php


echo “<form method=‘post’ action=‘esercizio5parte2.php’><input type=‘text’ name=‘parola’ />”;

echo “<input type=‘submit’ value=‘Invia’ /></form>”;


?>
```

Seconda parte: conteggio

```
<?php
```

```
//assegniamo alla variabile $testo la nostra poesia preferita

$testo = $nostra_poesia_preferita;



//stampiamo il conteggio della parola selezionata

echo substr_count($testo, $_POST[‘parola’]);



?>
```

Capitolo 17

```
<?php



$array1 = array(“a”, “b”, “c”);

$array2 = array(“d”, “e”, “f”);

$array3 = array(“g”, “h”, “i”);

$array = array_merge($array1, $array2, $array3);



$lunghezza = count($array);



for ($i=$lunghezza-1; $i>$lunghezza-3; $i—)
{

    unset($array[$i]);

}



foreach ($array as $el)
{

    echo “$el<br />”;

}
```

```
?>
```

Capitolo 19

Di seguito solo la parte da modificare per fare in modo che i post sul guestbook siano ordinati dal più recente al più vecchio.

```
//apriamo il guestbook in sola lettura per estrapolare i contenuti

$guestbook = fopen("$guestbook_testo","r");



//mettiamo i contenuti in una variabile

$contenuti = file_get_contents($guestbook_testo);



//chiudiamo il file

fclose($guestbook);



/*lo riapriamo in scrittura, in modalità w:
i vecchi contenuti sono memorizzati nella variabile */

$guestbook = fopen("$guestbook_testo","w");



/*dichiariamo una variabile $nuovi_contenuti: inseriamo i dati dell'ultimo messaggio

e aggiungiamo di seguito i vecchi contenuti */

$nuovi_contenuti = "<tr><td><b>$utente</b> scrive: <br /><br />

$messaggio<br /><br /><br />

<i>Messaggio scritto in data: $data_ora</i></td></tr>

".$contenuti;



//scriviamo il tutto sul guestbook
```

```
fwrite($guestbook,$nuovi_contenuti);



//chiudiamo

fclose($guestbook);
```

Capitolo 22

Prima parte, il form per inviare il post. Usiamo due campi testuali: il titolo e il testo del post. Per l'id dell'autore utilizziamo un campo nascosto, e inseriamo l'id dell'autore come attributo "value".

```
<form name='invio_post' method='post' action='invio-post.php'>

<input type='hidden' value='1' name='autore' />

<strong>Titolo</strong>: <input type='text' name='titolo' size='40'  /><br /><br />

<p style='text-align:center'><strong>Testo</strong></p>

<textarea name='testo' cols='70' rows='20'></textarea><br />

<input type='submit' name='invia_post' class='bottone' value='Invia!' />

</form>
```

Seconda parte, inseriamo i dati nel database. Utilizziamo la funzione "addslashes" per evitare che gli eventuali apici inseriti nel testo si confondano con gli apici che usiamo per costruire la query. Inseriamo la data in formato internazionale (Y-m-d H:i:s).

```
$query = "INSERT INTO Post (Id, IdAutore, Titolo, TestoHTML, Data) VALUES ('NULL',
'".$_POST['autore']."', '".$_POST['titolo']."', '".$_POST['testo']."', '".date('Y-m-d H:i:s')."')";

// Facciamo partire la query
mysql_query($query);
```

Capitolo 22

```
$query = 'SELECT p.Id, p.Titolo, p.TestoHTML, a.Nickname FROM Post p JOIN Autori a ON
```

```
p.IdAutore = a.IdAutore’;



$risultato = mysql_query($query);



while($record = mysql_fetch_array($risultato, MYSQL_ASSOC))

{

    echo “Post: “.$record[‘Titolo’].”<br />”;

    echo “Scritto da: “.$record[‘Nickname’].”<br />”;

    echo “<br />”;

    echo “Commenti… “;



    //Creiamo una query per i commenti

    $query_commenti = ‘SELECT * FROM Commenti WHERE IdPost = ‘.$record[‘Id’];

    $risultato_commenti = mysql_query($query_commenti);



    //E li stampiamo con un ciclo while

    while($record_commenti = mysql_fetch_array($risultato_commenti, MYSQL_ASSOC))

    {

        echo $record_commenti[‘Titolo’].”<br />”;

    }

    echo “<br />”;

}
```

Capitolo 23

```
SELECT SUM(NumCommenti) FROM Post WHERE NumCommenti != (SELECT MAX(NumCommenti) FROM Post)
```